TAMPEREEN TEKNILLINEN YLIOPISTO

LASSI HUHTALA

SENSORIVERKKOPOHJAINEN OMAISUUDENHALLINTAJÄRJESTELMÄ

Diplomityö

Tarkastajat: professori Tommi Mikkonen, professori Timo D. Hämäläinen
Tarkastajat ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 7. joulukuuta 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HUHTALA, LASSI: Sensoriverkkopohjainen omaisuudenhallintajärjestelmä
Diplomityö, 54 sivua, 11 liitesivua
Maaliskuu 2012
Pääaine: Ohjelmistotuotanto
Tarkastajat: professori Tommi Mikkonen, professori Timo D. Hämäläinen
Avainsanat: Langaton sensoriverkko, omaisuudenhallinta, paikantaminen, TUTWSN

Tässä työssä tutkitaan langattomien sensoriverkkojen avulla tapahtuvaa paikannusta, toteutetaan langatonta sensoriverkkoa hyödyntävä paikannussovellus ja arvioidaan sen hyötyjä. Sensoriverkkopohjaisen omaisuudenhallintajärjestelmän toteutusprojektissa on huomioitava paikannettavan esineen nopea etsiminen, oleellisen tiedon näyttäminen käyttäjälle ja käytetyimpien toimintojen yksinkertaisuus. Työn tavoitteena on selvittää, miten omaisuudenhallintajärjestelmä on mahdollista toteuttaa ja kuinka paljon hyötyä omaisuudenhallintajärjestelmän käyttöönottamisesta yritykselle on.

Langattomia sensoriverkkoja käytetään apuna monissa sovelluksissa, joissa tarkkaillaan ympäristön tapahtumia. Niiden avulla voidaan ohjata ja automatisoida teollisuuden prosesseja tai niitä voidaan käyttää apuna esineiden tai henkilöiden paikantamisessa. Verkossa oleva yksittäinen mittalaite voi sisältää useita eri antureita, jotka mittaavat erilaisia tapahtumia ympäristössä.

Työ jakaantuu kahteen osaan: Kirjallisuustutkimusosassa selvitetään langattomien sensoriverkkojen avulla suoritettavan paikannuksen teoriaa ja luodaan katsaus omaisuudenhallintajärjestelmien käyttökohteisiin. Toteutus- ja arviointiosassa toteutetaan paikannussovellus, joka käyttää langatonta sensoriverkkoa, ja arvioidaan sen soveltuvuutta yrityksen ensisijaisena omaisuudenhallintajärjestelmänä. Toteutuksessa on huomioitu käytettävyys, käyttäjäkeskeisyys ja ohjelmiston suunnittelu- ja toteutusvaiheessa on noudatettu ohjelmistoarkkitehtuurin standardeja.

Tutkimus osoittaa, että langattomia sensoriverkkoja pystytään hyödyntämään omaisuudenhallintajärjestelmissä, jotka käsittävät suuren alueen. Niiden avulla pystytään vähentämään kustannuksia, joita kuluu paikannettavien kohteiden etsimiseen laajoilta alueilta tai useita kerroksia käsittävän rakennuksen sisältä. Kohteen paikannusta varten kehitettyjen omaisuudenhallintajärjestelmien puutteellisuudesta johtuen toteutettiin omaisuudenhallintajärjestelmän prototyyppi, jonka avulla kohteiden paikannus on mahdollista. Paikannuksen tarkkuus riippuu sensoriverkkoon kytkettyjen mittalaitteiden määrästä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HUHTALA, LASSI: Wireless sensor network based asset management system
Master of Science Thesis, 54 pages, 11 Appendix pages
March 2012
Major: Software Engineering
Examiners: Professor Tommi Mikkonen, Professor Timo D. Hämäläinen
Keywords: Wireless sensor networks, asset management, localization, TUTWSN

This thesis examines the asset management with positioning application using wireless sensor networks. A part of this study is to implement an application that uses wireless sensor network for localization of moving objects. Positioning application's benefits are being evaluated and the application is being tested. Fast localization, displaying only relevant information in user interface and simplicity of the most used functions are important features in implementation project of a sensor network-based asset management system. The goal of this thesis is to determine how asset management system can be implemented and how much a company could benefit from the introduction of asset management system.

Wireless sensor networks are used in many applications where the events of environment are being monitored. They can be used to control and automate industrial processes or they can be used to locate items or people. A single measuring device on the wireless sensor network can include multiple sensors which can measure different events of the environment.

The thesis is divided into two parts: theory of the localization using wireless sensor networks is being explored in the literature study part. Literature section provides also an overview into asset management applications using wireless sensor networks. An implementation of a localization application is done in implementation and evaluating part of this thesis. The suitability of implemented application for the company's primary asset management system is being evaluated in evaluating part. Usability and user-centeredness are taken into account in the implementation phase of application's user interface. Design of the application's software architecture complies with the standards.

The study indicates that wireless sensor networks can be utilized in asset management systems which comprise a large area. Costs for searching items or people from large areas or buildings with many layers are reduced by using them. Due to lack of asset management systems which are using wireless sensor networks a prototype application was created. Accuracy of localization depends on the number of measurement units connected in sensor network.

ALKUSANAT

Vihdoin näyttäisi siltä, että valmistuminen diplomi-insinööriksi omalta kohdaltani on mahdollista. Työ valmistui muutaman kuukauden myöhässä omasta tavoitteestani, mutta siihen tuli lisää kohtia, joita en itse osannut heti huomioida. Monen vuoden opiskelun jälkeen tuntuu kuitenkin hienolta huomata, että on saanut jotain suoritettua valmiiksi asti.

Tämän diplomityön tarkoituksena on toimia teoksena, jota seuraamalla on mahdollista toteuttaa omaisuudenhallintajärjestelmä ja / tai jatkokehittää järjestelmää, joka tämän kirjallisuusosuuden sivussa on toteutettu. Työ on tehty vastaamaan Tampereen teknillisen yliopiston tietokonetekniikan laitoksen tarpeita. Työn sisällön olen yrittänyt pitää helposti ymmärrettävänä ja tekstin tukena olen käyttänyt paljon kuvia. Osa tämän työn kohdista vaatii lukijalta ohjelmoinnin ja ohjelmistotuotannossa käytettävien kaaviomallien perusteiden ymmärtämistä.

Haluan kiittää professori Tommi Mikkosta ja professori Timo D. Hämäläistä diplomityöni ohjaamisesta ja tarkastamisesta. Kiitokset kuuluvat myös tietokonetekniikan laitoksen henkilökunnalle työpisteen ja diplomityöaiheen tarjoamisesta. Teknisten asioiden selvittämisestä ja mittalaitteiden lainaamisesta kiitos kuuluu erityisesti Teemu Laukkariselle. Työn rahoituksesta kiitos kuuluu Tampereen teknillisen yliopiston tietokonetekniikan laitokselle.

Lisäksi kiitos kuuluu opiskelijaystävilleni, joiden kanssa tämän työn määrittelevät dokumentit ja pohjatyöt on tehty. Kiitos myös vanhemmilleni tarjoamastanne tuesta. Lopuksi haluan kiittää kaikkia ystäviäni: olette jaksaneet kysellä ja odotella valmistumistani ja se on kannustanut minua saattamaan tämän työn loppuun asti.

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

2D	Two-dimensional
Admin	Järjestelmän pääkäyttäjä.
AoA	Angle of Arrival
API	Application Programming Interface
C++	Bjarne Stroustrupin kehittämä ohjelmointikieli.
CSV	Comma-Separated Values
DLL	Dynamic-Link Library
DoA	Direction of Arrival
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IF	Infrared
IPS	Indoor Positioning System
MVC	Model View Controller
Node	Langaton mittauslaite, joka sisältää erityyppisiä sensoreita.
TDoA	Time Difference of Arrival
ToA	Time of Arrival
PC	Personal Computer
Qt	Ohjelmointiympäristö, jolla työhön liittyvä ohjelmisto on toteutettu.
RFID	Radio Frequency IDentification
RSS	Received Signal Strength
RSSI	Received Signal Strength Indication
TUTWSN	Langattomista sensoreista muodostettu langaton mittausverkko, joka on kehitetty TTY:n tietokonetekniikan laitoksella.
UML	Unified Modeling Language
US	Ultrasound
WLAN	Wireless Local Area Network

1 JOHDANTO

Omaisuudenhallintajärjestelmällä viitataan järjestelmään, jonka avulla esineitä ja asioita, joilla on arvoa yritykselle tai yksityiselle henkilölle, pystytään seuraamaan ja ylläpitämään. Seurattava esine voi olla esimerkiksi videotykki ja seurattavalla asialla voidaan tarkoittaa vaikkapa videotykin lampun jäljellä olevien käyttötuntien määrää. Esineitä voidaan paikantaa sisätiloissa liittämällä niihin sensori, joka lähettää tietoa sen sijainneista. Jotta esineen sijainti voidaan paikantaa siinä olevan sensorin avulla, tarvitaan sensoriverkko, joka käsittää useita kiinteissä paikoissa sijaitsevia sensoreita.

Suurissa rakennuskomplekseissa tai useita kerroksia käsittävässä rakennuksessa esineiden ja ihmisten paikantaminen huoneen tai edes kerroksen tarkkuudella on vaativa operaatio. Esimerkiksi Tampereen teknillisen yliopiston (TTY) kampus käsittää useita rakennuksia, joissa on useita kerroksia. TTY:llä liikkuu tuhansia eriarvoisia esineitä eri laitoksilla ja laitosten välillä. Jotkut näistä esineistä liikkuvat luonnostaan ja joidenkin on ehdottomasti oltava tiedetyssä paikassa. Esine voi olla esimerkiksi laboratoriomittauslaite, PC, tulostin tai videotykki.

Omaisuudenhallintajärjestelmän tarkoituksena on helpottaa erilaisten esineiden paikantamista reaaliaikaisesti, sekä tarjota mahdollisuus tutkia esineiden sijainteja menneinä ajankohtina. Lisäksi järjestelmän tulisi lähettää hälytys, jos jokin esine ei ole sille sallitulla alueella tai katoaa järjestelmästä kokonaan.

1.1 Tutkimuksen ja työn taustat

Omaisuudenhallintajärjestelmiä käytetään esineiden ja ihmisten paikallistamiseen, inventaarioiden ylläpitämiseen, omaisuuden analysointiin ja raporttien tekemiseen. Niitä käytetään tulostamaan viivakooditarroja tuotteille, joita luetaan viivakoodilukijalla, tai niiden avulla voidaan tallentaa esineen tietoja RFID-sirulle. Omaisuudenhallintajärjestelmiä käytetään myös raaka-ainehankitojen automatisoimiseen, kun järjestelmä ilmoittaa esimerkiksi sähköpostilla alihankkijoille, koska tarvittava raaka-aine on loppumassa.

Omaisuudenhallintajärjestelmän potentiaalisia käyttäjiä ovat erilaiset teollisuuden alan yritykset, joilla on laajat varastotilat ja paljon arvokkaita esineitä varastossaan. Muita mahdollisia käyttäjiä ovat turvallisuusalan yritykset, jotka haluavat saada nopeasti hälytyksen, jos omaisuutta katoaa tarkkailun kohteena olevalta alueelta. Myös

logistiikka-alan yritykset haluavat tietää, missä heidän kulkuneuvonsa ja kyydissä oleva tavara sijaitsee tiettynä ajanhetkenä.

Tässä työssä tutkitaan pääasiassa omaisuuden paikantamiseen liittyviä teorioita ja tuloksia. Tutkimuksen kohteena on selvittää, kuinka langattomat sensoriverkot toimivat osana omaisuudenhallintajärjestelmää ja kuinka niitä hyödyntävä sovellus on mahdollista totetuttaa. Toteutettua ratkaisua verrataan olemassa oleviin omaisuudenhallintajärjestelmiin.

Tutkimustuloksien perusteella päädyttiin tekemään kokonaan oma käyttöliittymän esittelyversio omaisuudenhallintajärjestelmälle, koska vuoden 2010 tammikuussa ei löytynyt yhtään avoimen lähdekoodin sovellusta, joka olisi sopinut TTY:n käyttötarkoituksiin. Työ aloitettiin kandidaatintyönä TTY:n tietokonetekniikan laitokselle. Samalla selvisi, että Qt olisi paras kehitysympäristö toteuttaa järjestelmä. Käyttöliittymä päivittyi kahdeksi eri sovellukseksi, joilla on toisistaan eroavat käyttöliittymät, ja jotka osaavat hyödyntää TTY:llä käytössä olevaa TUTWSN-mittausverkkoa [1].

Ensiksi toteutettiin editointityökalu (pääkäyttäjän sovellus), jolla asetetaan karttapohjille huoneet, alueet ja kiinteät mittalaitteet (nodet). Sen avulla liitetään esineet ja mittalaitteet toisiinsa, jotta esineen paikannus olisi mahdollista. Editointityökalun jälkeen toteutettiin varsinainen monitorointisovellus (peruskäyttäjän sovellus), jonka avulla TTY:n alueella liikkuvia esineitä on mahdollista paikantaa reaaliaikaisesti.

1.2 Työn tavoitteet ja aiheen rajaus

Työn päätavoite on luoda järjestelmän käyttöliittymästä mahdollisimman selkeä ja helppokäyttöinen. Järjestelmän käyttöliittymän tärkein tehtävä on esittää paikannettavan esineen sijainti karttapohjan päällä. Toteutuksen tuloksena syntynyttä järjestelmän käyttöliittymää käyttämällä selvitetään, kuinka nopeasti eri toiminnot on mahdollista suorittaa järjestelmän avulla. Järjestelmän käyttöliittymää testataan eri tapauksissa, joiden avulla tulokset ja arvio järjestelmän soveltuvuudesta omaisuudenhallintajärjestelmäksi on esitelty myöhemmin tässä työssä. Edellä kuvattuja ominaisuuksia mitataan järjestelmän suorituskyvyn ja käytettävyyden suhteen.

Koska langattomien sensoriverkkojen taustalla toimivat järjestelmät käsittävät paljon teoriaa ja eri paikannusmenetelmiä on useita, tässä työssä tarkastellaan pääasiassa TTY:n tietokonetekniikan laitoksella kehitettyä sensoriverkkoa ja tuloksena syntyneen järjestelmän ohjelmistoratkaisuja. Työn rajaukseen kuuluu lisäksi tulosten analysointi, jossa esimerkiksi tarkastellaan, kuinka tarkasti esine pystytään paikantamaan omaisuudenhallintajärjestelmän avulla.

1.3 Diplomityön rakenne

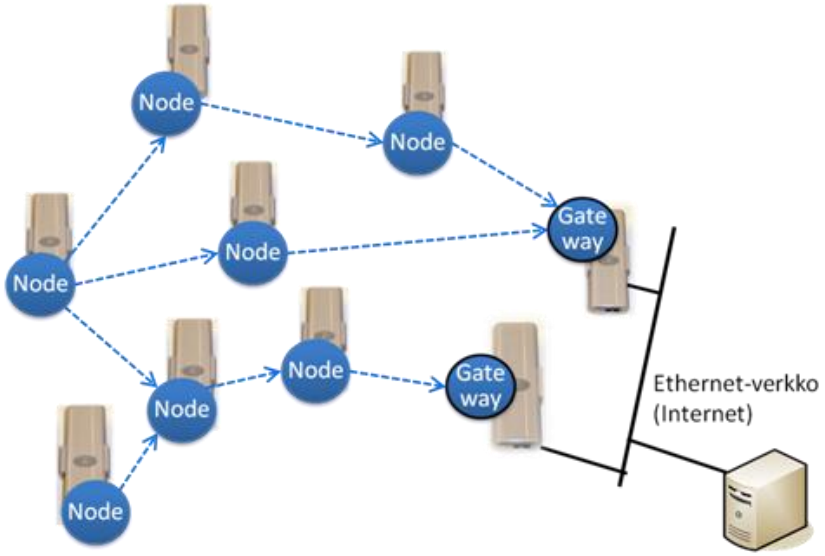
Luku kaksi muodostaa työn teoriaosuuden, jossa luodaan katsaus langattomien sensoriverkkojen perusteisiin ja käyttökohteisiin. Siinä tarkastellaan myös langattomien sensoriverkkojen käyttöä omaisuudenhallintajärjestelmissä ja tutustutaan erilaisiin paikannustekniikoihin, joita on mahdollista toteuttaa langattomien sensoriverkkojen avulla. Luvut 3-5 muodostavat työn toteutusosuuden. Niissä esitellään toteutustyönä syntyneen omaisuudenhallintajärjestelmän rakennetta ja tekniikoita, joita sen ohjelmoinnissa on käytetty. Luvuissa neljä ja viisi selvitetään lisäksi erään mahdollisen omaisuudenhallintajärjestelmän ohjelmistoratkaisuja. Luvussa kuusi on arvioitu järjestelmän toteutuksen onnistumista ja sen soveltuvuutta käyttökohteeseensa. Siinä selvitetään myös, kuinka paljon hyötyä paikannusjärjestelmän käyttöönottamisesta yritykselle on. Luvussa seitsemän esitellään yhteenveto työn tuloksista, sekä arvio projektin onnistumisesta.

2 LANGATTOMAT SENSORIVERKOT JA KÄYTTÖKOHTEET

Langattomat sensoriverkot koostuvat useista mittalaitteista, jotka välittävät tietoa keskenään. Verkossa oleva mittalaite voi sisältää useita antureita, jotka mittaavat muun muassa lämpötilaa, ilmankosteutta, hiilidioksidipitoisuutta, äänenvoimakkuutta tai sensorin ohi liikkuvien kohteiden määrää. Langattomia sensoriverkkoja käytetään esimerkiksi paikannus- ja tarkkailusovelluksissa.

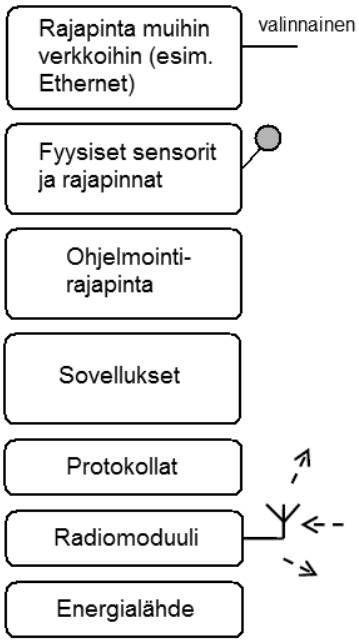
2.1 Langattomien sensoriverkkojen perusidea

Kuvassa 1 on esitetty, kuinka mittalaitteiden lähettämät viestit kulkevat verkossa. Sensoriverkon mittalaite lähettää viestin naapurilaitteelleen, joka edelleen välittää viestin seuraavalle, kunnes viesti saavuttaa sensoriverkon yhdyspisteen (gateway). Yhdyspisteestä on yhteys ethernet-verkkoon, josta viesti välitetään palvelimelle ja tietokantaan.



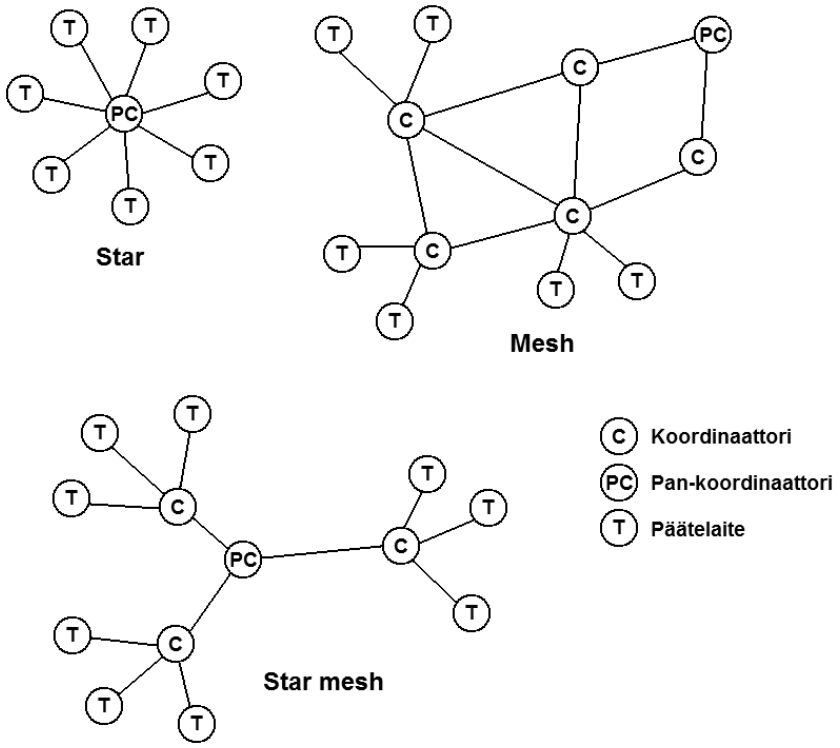
Kuva 1: Langattoman sensoriverkon yksinkertaistettu rakenne [1, 2, 3].

Yksinkertaisimmillaan sensoriverkon mittalaite sisältää antureita, radiomoduulin ja energialähteen. Anturit mittaavat sen ympärillä tapahtuvia ilmiöitä (esimerkiksi lämpötilan vaihtelu). Radiomoduuli hoitaa näytteistämisen ja lähettämisen. Energialähteenä voi toimia vaikkapa kaksi AA-paristoa [4]. Kuvassa 2 on esitetty mittalaitteen koko rakenne.



Kuva 2: Mittalaitteen rakenne.

Langattomat sensorit voivat toimia itsenäisinä laitteina tai niiden toimintaa voi ohjata yksi tai useampi keskuslaite riippuen verkon topologiasta [5, 6]. Erilaisia langattomien sensoriverkkojen topologiota ovat esimerkiksi tähtimäinen verkkorakenne (star), reitittävä verkko (mesh) ja näiden kahden yhdistelmä, eli star mesh (Kuva 3). Tähtimäisessä topologiassa verkkoa hallitsee yksi laite, jota kutsutaan PAN-koordinaattoriksi [5, 6]. Mesh verkossa tukiaseman vaihtaminen ja verkossa liikennöiminen on mahdollista ilman yhteyden katkeamista.



Kuva 3: Langattomien sensoriverkkojen topologioita.

2.2 Langattomien sensoriverkkojen käyttökohteita

Langattomia sensoriverkkoja käytetään esimerkiksi teollisuudessa, maanviljelyn edistämiseen, terveydenhuollon apuna ja paikannussovelluksissa. Yhteistä kohteille on vaikeat olosuhteet luoda langallinen verkko tai langallisen verkon aiheuttamat korkeat kustannukset suhteessa saavutettuun hyötyyn. Niiden avulla mitataan ympäristön tapahtumia ja säädetään järjestelmää olosuhteiden mukaan. Esimerkiksi kasvihuoneen kastelujärjestelmään liitetty langaton sensoriverkko mittaa lämpötilaa, sekä ilmankosteutta, ja säätää automaattista kastelujärjestelmää ympäristöstä mitattujen parametrien mukaan [7, 8, 9, 10, 11, 12].

Ulkotiloissa langattomilla sensoriverkoilla voidaan kattaa pienellä määrällä laitteita laaja alue ympäristön monitorointia varten. Parhaimmillaan langattomien sensoriverkkojen mittalaitteet kestävät äärimmäisiä olosuhteita (-40 °C), pysytyvät lähettämään signaalin yli kilometrin päähän ja pystyvät toimimaan puoli vuotta samoilla paristoilla [13]. Edellä mainittujen ominaisuuksien vuoksi langattomat sensoriverkot ovat paras vaihtoehto sellaisiin kohteisiin, joihin langallista verkkoa ei voida pitkien välimatkojen tai vaikeakulkuisen maaston vuoksi voida pystyttää.

2.3 Sensoriverkkojen käyttö omaisuudenhallintasovelluksissa

Omaisuudenhallinnalla tarkoitetaan systemaattista prosessia olemassa olevan omaisuuden operoinnilla, ylläpitämisellä ja edistämisellä. Sen tarkoituksena on helpottaa tiedon saatavuutta omaisuuden olinpaikasta, määrästä ja laadusta [14, 15, 16]. Omaisuudenhallintajärjestelmiä käyttämällä pyritään laskemaan esineiden tai ihmisten etsimiseen kuluvia kustannuksia. Järjestelmien avulla yritetään myös saavuttaa mahdollisimman korkea automaation aste [17].

Passiivisia RFID-siruja käytetään monissa omaisuudenhallintajärjestelmissä. Ne ovat yleisesti käytettyjä, koska niiden etuina ovat matalat kustannukset, helppo asennus, matala virrankulutus, pitkä elinikä ja suuri verkon koko [6, 17, 18]. Passiiviset RFID-sirut vaativat tuekseen RFID-lukijan ja isossa RFID-verkossa lukijoita tarvitaan paljon. Luvuissa 3 ja 4 esiteltävä järjestelmä käyttää TUTWSN-verkkoa. Sen etuja RFID-siruja käyttäviin järjestelmiin verrattuna ovat mittalaitteen toimintojen määrä, verkon itsenäinen toiminta, verkon kantama, turvallisuus ja säädettävyys käyttökohteessa [1, 2, 3, 6].

2.4 Paikannus langattomien sensoriverkkojen avulla

Langattomia sensoriverkkoja käytetään sisä- ja ulkotiloissa kohteiden paikantamiseen. Sisätiloissa tapahtuvaan paikannukseen ne soveltuvat erityisen hyvin, koska tarkka GPS-paikannus ei ole mahdollista sisätiloissa [19]. Langattomien sensoriverkkojen avulla tapahtuva paikannus suoritetaan yleensä verkossa olevien kiinteiden ja liikkuvien laitteiden kommunikoimisen avulla. Kiinteiden laitteiden sijainti tunnetaan ja liikkuvien laitteiden sijainti lasketaan suhteessa kiinteisiin laitteisiin. Liikkuvien laitteiden sijainti voidaan laskea esimerkiksi kiinteiden laitteiden lähettämistä signaalien voimakkuuksista tai saapuneiden signaalien välisistä aikaeroista.

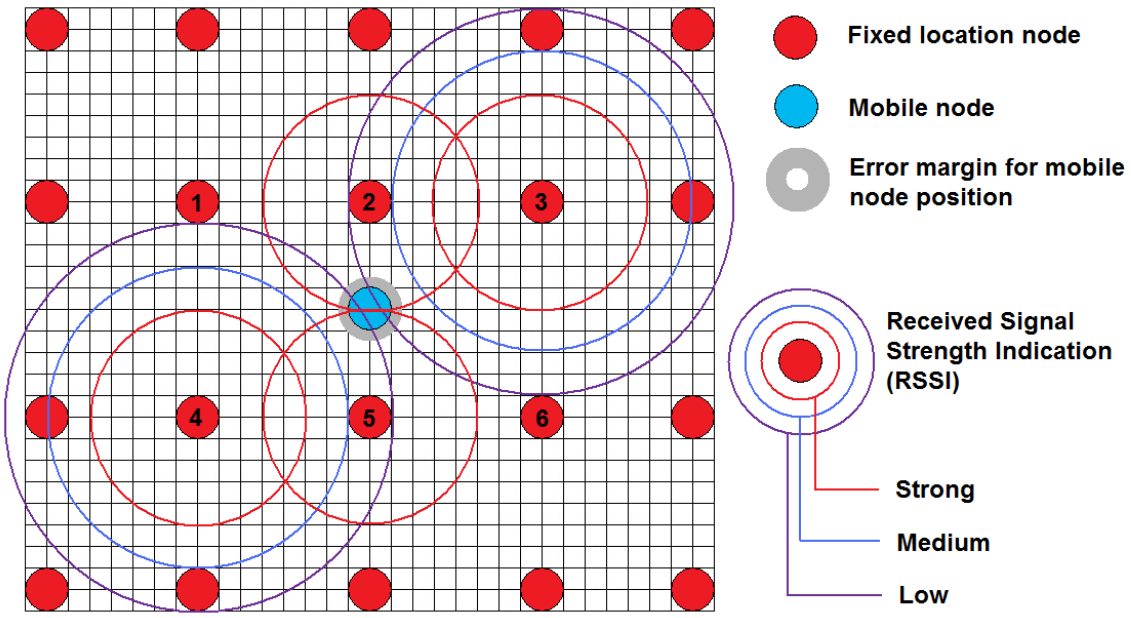
Langattomia sensoriverkkoja hyödyntämällä voi paikannuksen tarkkuus olla parhaimmillaan muutamien senttimetrien luokkaa, kun GPS-paikannus tai TDoA-menetelmään perustuvan paikannuksen tarkkuus voi erota oikeasta sijainnista useita kymmeniä metrejä [18]. Langattomien sensoriverkkojen avulla tapahtuvan paikannuksen tarkkuus riippuu myös sensoriverkon koosta: mitä enemmän kiinteitä laitteita, sitä tarkempi paikannustulos.

Yleisimmin sisätilojen paikannusjärjestelmissä (engl. *Indoor Positioning System*, IPS) tutkittuja ja käytettyjä langattomia tekniikoita ovat muun muassa RFID, Bluetooth, ZigBee, ultraäänet (US), infrapuna (IR) ja langaton lähiverkkotekniikka (WLAN tai WiFi) [18]. Seuraavassa on esitelty yleisiä langattomien sensoriverkkojen paikannustekniikoita tarkemmin.

2.4.1 RSS

Eniten langattomien sensoriverkkojen paikannusjärjestelmissä käytetty tekniikka perustuu vastaanotettujen signaalien voimakkuuksien tutkimiseen. Se perustuu etäisyyden arviointiin. Signaalien voimakkuuksiin perustuvassa tekniikassa tarvitaan kiinteitä laitteita, joiden sijainti tiedetään. RSSI-arvo on ominaisuus, joka perustuu standardiin useissa radiolaitteissa, esimerkiksi ZigBee. Mittaukset voidaan suorittaa joko tutkimalla paikannettavan laitteen mittaamia kiinteiden laitteiden signaalien voimakkuuksia, tai kiinteiden laitteiden mittaamia paikannettavan laitteen signaalin voimakkuutta [18, 20, 21, 22].

RSSI-arvojen tutkimiseen perustuvan paikannustekniikan rajoituksia ovat RSSI-arvon epävakaus ja muutokset ajan kuluessa. Menetelmän tarkkuus on muutamien metrien luokkaa [18, 21, 23]. Kuvassa 4 on esimerkki RSSI-arvoihin perustuvasta paikannuksesta.



Kuva 4: RSSI-arvoihin perustuva paikannusmenetelmä. Keskellä olevan liikkuvan mittalaitteen sijainti lasketaan ympärillä olevien kiinteiden laitteiden lähettämien signaalien voimakkuuksien perusteella. Voimakkaimmat vastaanotetut signaalit tulevat laitteilta numero 2 ja numero 5, joten liikkuvan mittalaitteen todennäköinen sijainti on niiden kahden välissä.

2.4.2 AoA

Angle of Arrival-tekniikka (kutsutaan myös nimellä Direction of Arrival (DoA)) perustuu vaiheen tai amplitudin muutokseen, joka johtuu antennin suuntauksesta. AoA-tekniikkaa käytetään yleensä ultraääni- ja radiotaajuussignaalien kanssa. Sitä käytetään usein RSS- ja ToA-tekniikoiden tukena paikannusmittauksissa. AoA-tekniikan tarkkuus on muutamia metrejä. [18, 21, 22]

AoA-tekniikkaa käytetään usein mikrofonijoukon avulla. Se perustuu arvioon vastaanotettujen äänisignaalien suunnasta. Sitä käytetään esimerkiksi sovelluksissa, joissa videokameran halutaan kääntyvän automaattisesti puhujan suuntaan. Heikkoutena on äänen heijastuminen esineistä ja seinistä, eli ääni ei aina tule suoraan lähteeltä vastaanottimelle [24].

2.4.3 ToA ja TDoA

Time of Arrival-tekniikka perustuu valonnopeudella eteneviin sähkömagneettisiin signaaleihin. Tekniikan tueksi tarvitaan erittäin tarkkojen kellojen käyttöä. Etäisyys lähettimen ja vastaanottimen välillä voidaan laskea käyttämällä ToA-tekniikkaa ja tiedossa olevaa signaalin nopeutta. Sekä lähettimen, että vastaanottimen päässä käytetään aikaleimoja. ToA-tekniikan (kutsutaan myös nimellä Time of Flight (ToF)) tarkkuus on alle metrin luokkaa, mutta jo yhden mikrosekunnin synkronointivirhe voi johtaa 300 metrin virheeseen paikannuksessa [18, 21, 22].

Saman signaalin saapumisen aikaeroa kahdelle eri sensorille kutsutaan nimellä Time Difference of Arrival (kutsutaan myös nimellä Time Difference of Flight (TDoF)) [25, 26]. TDoA-tekniikalla toteutettu mittaus ei ole riippuvainen lähettävän sensorin kellon poikkeamasta. TDoA-tekniikkaa käytetään esimerkiksi matkapuhelimien paikallistamiseen [22].

3 OMAISUUDENHALLINTASOVELLUKSEN VAATIMUKSET JA ARKKITEHTUURI

Tässä luvussa kuvataan aluksi ongelmat ja tarpeet, joiden pohjalta omaisuudenhallintajärjestelmää aloitettiin kehittämään. Ongelman kuvauksen jälkeen esitellään järjestelmälle asetettuja vaatimuksia käyttöliittymän ja tiedostomuotojen suhteen. Lopuksi kuvataan järjestelmän toimintoja ja tekniikoita ja miten ratkaisuihin on päädytty. Ratkaisujen tukena esitellään pätkiä ohjelmakoodista ja UML-kaavioita.

3.1 Ongelman kuvaus ja järjestelmälle asetetut vaatimukset

Tietokonetekniikan laitoksen kehittämästä TUTWSN-mittausverkkosta saatavaa dataa voidaan käyttää hyödyksi monella tavalla. Verkon mittalaitteiden sisältämät anturit mittaavat muun muassa lämpötilaa, ilman hiilidioksidipitoisuutta ja valoisuusarvoa. Työn kannalta tärkein TUTWSN:n taltioima informaatio on mittalaitteiden rekisteröimä tieto sen lähellä olevista naapurilaitteista ja naapurilaitteiden lähettämä RSSI-arvo. Tutkimalla esineeseen liitetyn liikkuvan mittalaitteen tietoja sen naapurilaitteista ja niiden RSSI-arvoista, voidaan esineen sijainti TUTWSN-verkon sisällä paikantaa.

Ennen kuin paikannettavan esineen tai mittalaitteen sijainti voidaan graafisesti esittää, täytyy järjestelmälle kertoa kiinteiden laitteiden sijainnit, alueiden ja huoneiden sijainnit ja esineet täytyy yhdistää jotenkin liikkuvien mittalaitteiden kanssa, jotta esine lähettäisi ja vastaanottaisi tietoa muilta sensoriverkon mittalaitteilta. Koska järjestelmä kattaa monia rakennuksia ja kerroksia, on ne eriteltävä käyttöliittymässä. Kerroksessa ja yhdessä huoneessa saattaa olla useita mittalaitteita yhtä aikaa. Samassa huoneessa sijaitsevien mittalaitteiden sijainti tulee pystyä esittämään järkevästi käyttöliittymänäkymässä.

Järjestelmä sisältää useita kohteita, joita pystytään etsimään hakusanalla. Järjestelmässä voi olla useita samannimisiä paikannettavia esineitä. Esineet täytyy pystyä tunnistamaan muutenkin, kuin nimen perusteella. Ensisijaisesti halutaan tietää, missä esine on juuri nyt, mutta esineiden sijaintitietoa voidaan tarkastella myös menneinä ajankohtina. Tarkastelemalla esineen aikaisempia sijainteja voidaan ennustaa, missä esineen sijainti tulee olemaan lähitulevaisuudessa.

Järjestelmän toteutukselle asetetut vaatimukset tärkeysjärjestyksessä ovat esiteltynä seuraavassa:

Ylläpitäjä (Admin):

1. Sisäänkirjautuminen järjestelmään.
2. Karttojen ja alueiden tuonti ja vienti csv-tiedostoon.
3. Karttapohjien lataus, alueiden ja mittalaitteiden sijaintien lataaminen pääkäyttäjän näkymään.
4. Alueen (huoneen) määrittäminen karttapohjalle piirtämällä ja alueen poistaminen.
5. Kiinteiden ja liikkuvien mittalaitteiden tuonti pääkäyttäjän sovellukseen csv-tiedostosta.
6. Mittalaitteen yhdistäminen alueeseen ja poistaminen alueelta ja asettelu karttapohjalla.
7. Esineiden tuonti järjestelmään csv-tiedostosta.
8. Liikkuvan mittalaitteen ja esineen yhdistäminen.
9. Tallennus ja lataus kaikille muokatuille karttapohjille, mittalaitteiden ja alueiden sijainneille ja esineiden ja mittalaitteiden yhdistelyille.
10. Avaa oletuksena viimeiseksi muokatun projektin.

Peruskäyttäjä:

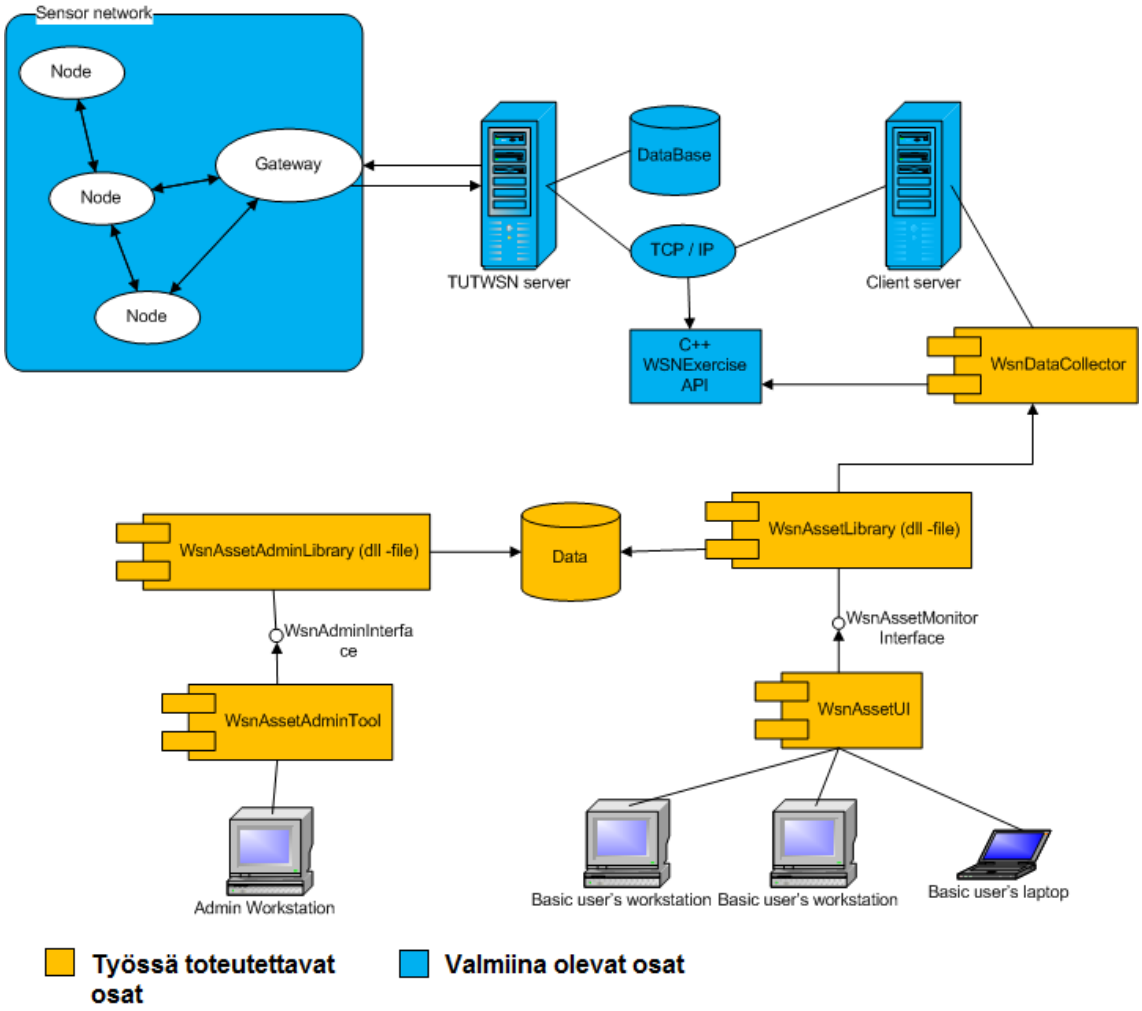
1. Koko kiinteistön tms. karttojen, mittalaitetietojen ja alueiden avaaminen monitorointinäkymään.
2. Karttapohjan loitontaminen ja lähentäminen.
3. Välilehtiä napsauttamalla tapahtuva usean karttapohjan (kerrosten / rakennusten) välillä navigointi.
4. Hakutoimintojen näyttäminen.
5. Liikkuvan mittalaitteen viimeisimmän paikkatiedon hakeminen ja laitteen näyttäminen karttapohjalla viimeisimmän sijainnin mukaan.
6. Alkuasetelmassa kaikkien liikkuvien mittalaitteiden näyttäminen.
7. Alueiden / tilojen ja mittalaitteiden / esineiden värikoodaus.
8. Alueiden tunnisteiden näyttäminen / piilottaminen karttapohjalla.
9. Kiinteiden mittalaitteiden näyttäminen / piilottaminen karttapohjalla.
10. Hälytä.

3.2 Omaisuudenhallintajärjestelmän arkkitehtuuri

Omaisuudenhallintajärjestelmää käyttävälle henkilölle tärkein osuus on järjestelmän käyttöliittymä. Pääkäyttäjän ja peruskäyttäjän käyttöliittymän lisäksi järjestelmään kuuluu molemmille sovelluksille omat logiikkamoottorit, jotka käsittelevät muun muassa sovelluksen tietorakenteiden hallinnan ja csv-tiedostojen lukemisen.

Toteutuksen tuloksena syntyi kaksiosainen omaisuudenhallintajärjestelmä, joka koostuu pää- ja peruskäyttäjän sovelluksista. Järjestelmän muunneltavuuden ja tietoturvallisuuden säilyttämisen kannalta tehtiin kaksi erillistä sovellusta. Lisäksi molemmilla sovelluksilla on eri käyttötarkoitus. Kummallakin sovelluksella on toisistaan eroava käyttöliittymä ja logiikkamoottori, mutta osa funktioista toimii samalla tavalla molemmissa sovelluksissa. Peruskäyttäjän sovellus tarvitsee lisäksi käyttöönsä mittalaitteiden mittausdataa keräävän komponentin, jonka avulla muutetaan TUTWSN-mittausverkosta saatava data omaisuudenhallintajärjestelmälle soveltuvaan muotoon. Järjestelmän kokonaisarkkitehtuurin kuvaus on esitetty kuvassa 5.

Kuvassa 5 sinisellä värillä merkityt osat kuvaavat valmiiksi olemassa olevia osia järjestelmästä ja oranssilla merkityt komponentit toteutetaan tässä työssä. Kuvassa vasemmalla ylhäällä oleva kokonaisuus kuvaa TUTWSN-sensoriverkkoa. TUTWSN-verkko lähettää mittalaitteiden mittaaman datan TUTWSN-palvelimen kautta tietokantaan ja TCP/IP-yhteyden läpi asiakaspalvelimelle (client server) *WSNExerciseAPI*-rajapintaa hyödyntämällä. Asiakaspalvelimella on ajossa *WsnDataCollector*-sovellus, joka suodattaa TUTWSN-verkon mittausdatan peruskäyttäjän sovellukselle (*WsnAssetLibrary* + *WsnAssetUI*) sopivaan muotoon. Tässä työssä asiakaspalvelimena toimii TTY:n lintulan palvelintietokone (mustatilhi) [27]. Pääkäyttäjän sovellus (*WsnAssetAdminLibrary* + *WsnAssetAdminTool*) eivät tarvitse toimiakseen muuta, kuin sensoriverkon kattavan alueen pohjapiirrustukset digitaalisessa muodossa (esimerkiksi jpg- tai png-formaatti). Kuvan keskellä oleva oranssi ”*Data*”-tietosäiliö kuvaa kaikkia pää- ja peruskäyttäjän sovelluksen käyttämiä pohjapiirrustuksia ja csv-tiedostoja.



Kuva 5: Järjestelmän kokonaisarkkitehtuurin kuvaus.

3.3 Toteutuksessa käytetyt työkalut ja tekniikat

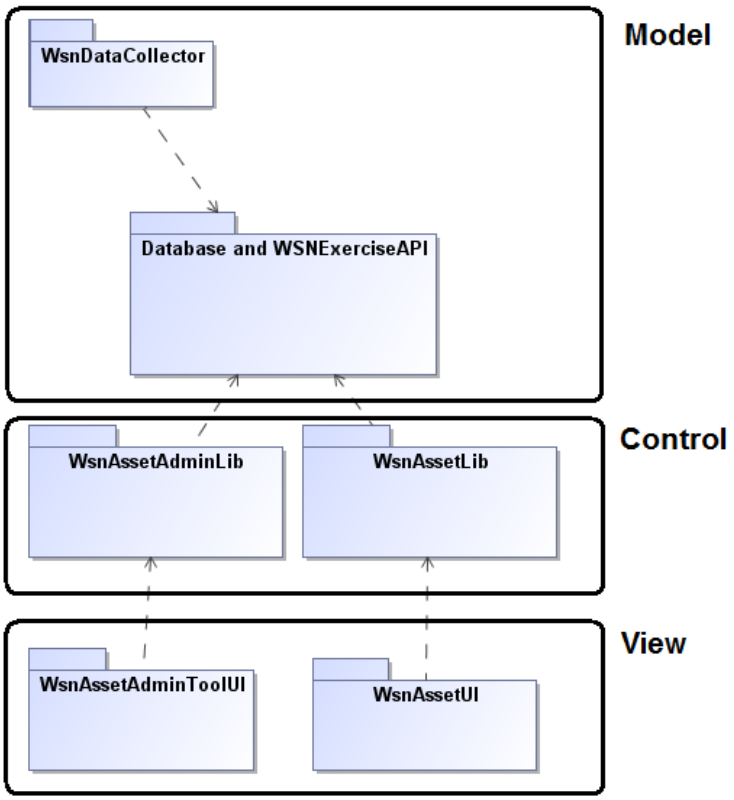
Järjestelmän suunnittelussa on käytetty ohjelmistoarkkitehtuurien mukaisia suunnittelumalleja. Luokka- ja sekvenssikaavioiden kuvauksessa käytettiin UML-notaatiota [28]. Käyttöliittymien suunnittelussa on otettu huomioon käytettävyys ja käyttäjäkeskeisyys. Työn pohjana ovat toimineet aikaisemmin TTY:n tietokonetekniikan laitokselle tehty kandidaatintyö [15] ja omaisuudenhallintajärjestelmän määrittelydokumentti [16].

Pää- ja peruskäyttäjän sovelluksien toteuttamisessa käytettiin Qt:ta [29]. Käyttöliittymäosuudet toteutettiin lähes kokonaan Qt Designer-työkalun avulla ja logiikkamoottorit Qt:n C++ -luokkakirjastoja hyödyntämällä. Moottoreista luotiin dll-kirjastotiedostot, jotka tarjoavat rajapinnat (liite 1 ja liite 2) käyttöliittymäpuolen komponenteille. Reaaliaikaisen datan hakemista varten toteutettiin pieni sovellus C++:lla (kuva 5, *WsnDataCollector*), joka hakee TUTWSN-verkon palvelimelta *WSNExerciseAPI*-rajapinnan [30] kautta tulevaa dataa ja kirjoittaa sen csv-tiedostoon.

Pääkäyttäjän sovellusta käytettiin TTY:n rakennusten karttapohjien, huoneiden ja TUTWSN-mittausverkon mittalaitteiden sijaintien viemisessä csv-tiedostoon. Paikannuksessa käytetään TUTWSN-mittausverkosta saatavaa dataa ja pääkäyttäjän sovelluksen avulla tallennettujen csv-tiedostojen sisältämän datan kombinaatiota. Paikannetut kohteet näytetään peruskäyttäjän käyttöliittymän karttapohjalla, kohteen oikeaa fyysistä sijaintia vastaavassa kohdassa.

3.4 Järjestelmän yleiset ohjelmistoratkaisut

Järjestelmän toteuttaminen jakautui kahteen isoon kokonaisuuteen: pääkäyttäjän sovellukseen ja peruskäyttäjän sovellukseen. Kuvassa 6 on esitetty korkean tason kuvaus järjestelmän MVC-arkkitehtuurista. Vasemmalla puolella kuvaa olevat komponentit: *WsnAssetAdminLib* ja *WsnAssetAdminToolUI* muodostavat yhdessä pääkäyttäjän sovelluksen ja kuvassa oikealla puolella olevat *WsnAssetLib* ja *WsnAssetUI* muodostavat peruskäyttäjän sovelluksen. Kuvan keskellä oleva komponentti (*Database and WSNExerciseAPI*) kuvaa tietokantaa, sekä TUTWSN-mittausverkkoon liittyvää C++ -ohjelmointirajapintaa, joiden toteutukseen ei tässä työssä oteta kantaa. Kuvan 6 ylimpänä oleva komponentti (*WsnDataColletor*) on itsenäinen sovellus, joka lukee *WSNExerciseAPI*:n kautta tulevan datan ja tallentaa sen taulukkomuotoon (csv-tiedostoon), peruskäyttäjän sovelluksen tarpeita varten.



Kuva 6: Korkean tason MVC-kuvaus järjestelmän komponenteista.

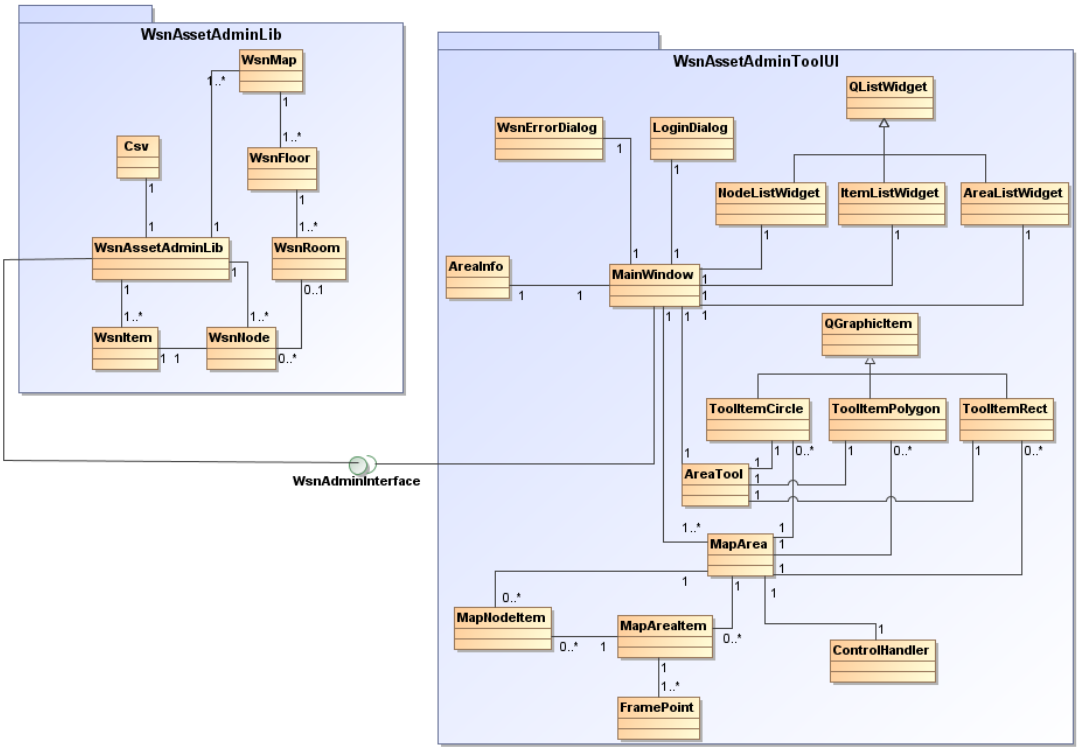
Pää- ja peruskäyttäjän sovelluksien moottorit päätettiin toteuttaa dynaamisesti linkitettävinä kirjastoina. Dynaamisesti linkitettävät kirjastot ovat käyttökelpoisimpia suurissa sovelluksissa, ja jos järjestelmä koostuu useista ohjelmista, jotka tarvitsevat samankaltaisia rutiineja [31]. Dynaamisten kirjastojen käytöstä on muitakin etuja: sovelluksen päivittäminen ei vaadi koko ohjelmiston uusimista, korvataan vain vanha dynaamisesti ladattava kirjasto uudella versiolla. Vaihtoehtoisesti voidaan toteuttaa ulkoasultaan erilaisia käyttöliittymäkomponentteja, jotka käyttävät samaa dynaamisesti ladattavaa kirjastoa eri tavoin, luomalla esimerkiksi komentorivipohjainen käyttöliittymä ja graafinen, hiirellä ohjattava käyttöliittymä. Molemmat käyttöliittymät kutsuvat dynaamisesti ladattavan kirjaston funktioita samalla tavalla, mutta lopputulos näytetään käyttäjälle eri tavalla.

Järjestelmän käyttöliittymät tehtiin pääosin Qt Designerilla, joka mahdollistaa käyttöliittymäelementtien nopean asettelun valikosta sovellukseen raahaa ja pudota -toiminnon avulla. Qt Designer mahdollistaa myös erilaisten sommitelmien asettamisen, jonka avulla käyttöliittymän elementit asetellaan käyttöliittymän näkymässä mielenkiintoiseen ja helposti luettavaan muotoon. Osa käyttöliittymien elementeistä ohjelmoitiin ilman Qt Designer-työkalua, esimerkiksi *Node*-, *Item*- ja *AreaListWidget* (kuva 7) kehitettiin käyttämällä ainoastaan Qt:n tekstieditoria.

4 PÄÄKÄYTTÄJÄN SOVELLUS

Pääkäyttäjän sovelluksen tarkoituksena on pystyä siirtämään sensoriverkon fyysinen ympäristö päätelaitteessa näytettävään käyttöliittymään. Pääkäyttäjän käyttöliittymän avulla ladataan karttapohjat, joiden alueella sensoriverkko sijaitsee ja sen avulla luodaan rakennukset ja kerrokset. Karttapohjien lataamisen jälkeen niille sijoitellaan huoneet ja kiinteät mittalaitteet vastaamaan niiden sijaintia reaalimaailmassa. Lopuksi pääkäyttäjän sovelluksella yhdistetään esineet liikkuvien mittalaitteiden kanssa.

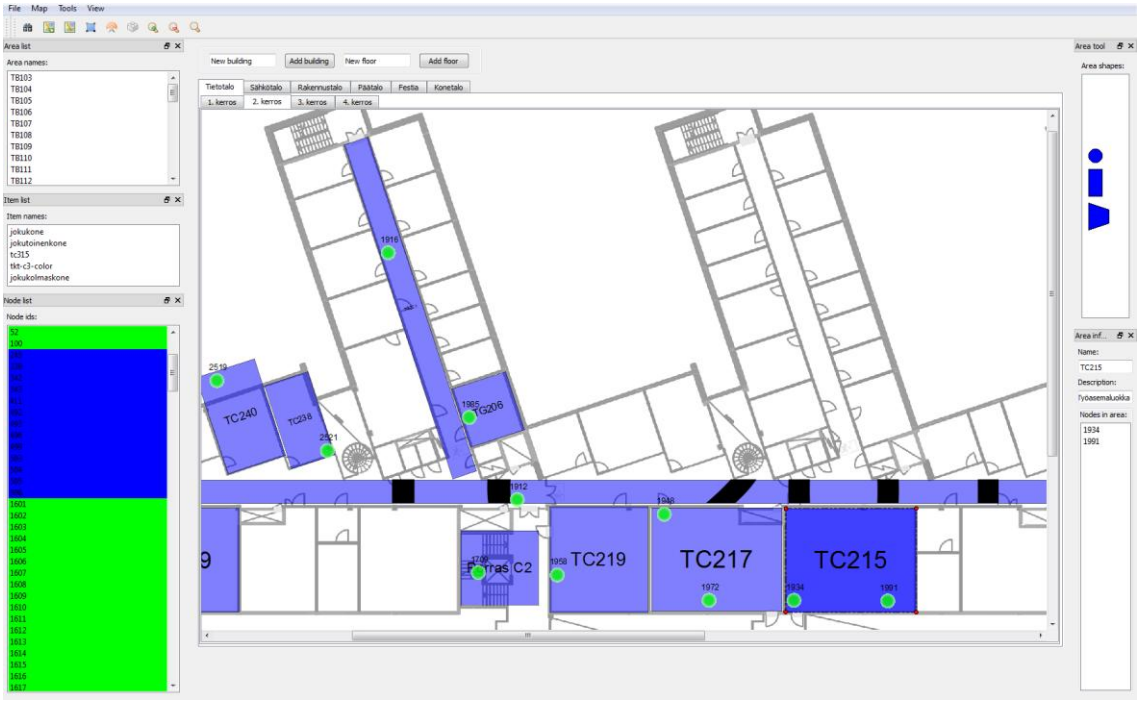
Pääkäyttäjän sovellus koostuu logiikkamoottorista ja sen tarjoamasta rajapinnasta, sekä käyttöliittymästä. Kuvassa 7 on esitetty pääkäyttäjän sovelluksen rakenne komponentti- ja oliotasolla. Pääkäyttäjän sovellus lukee csv-tiedostoihin tallennettua tietoa alueista, mittalaitteista ja esineistä. Huoneiden ja mittalaitteiden sijainteja voidaan muokata myös suoraan editoimalla csv-tiedostoja, joihin ne on tallennettu. Pääkäyttäjän sovelluksen käyttämät csv-tiedostot ja esimerkit niiden sisällöstä on lueteltu tämän työn liitteissä 3-7.



Kuva 7: Pääkäyttäjän sovelluksen moottorin ja käyttöliittymän kuvaus luokkakaaviona.

Pääkäyttäjän sovelluksen avulla valitaan karttapohjat sensoriverkon kattaville alueille, luodaan alueet ja huoneet, sijoitellaan kiinteät mittalaitteet karttapohjalle niiden fyysistä

sijaintia vastaaville kohdille ja liitetään esineet paikannettaviin mittalaitteisiin. Kuvassa 8 on esitetty kuvankaappaus pääkäyttäjän käyttöliittymänäkymästä. Kohdassa 4.1 kuvataan pääkäyttäjän sovelluksen moottorin toimintaa ja sen sisältämiä olioita. Kohdissa 4.2 ja 4.3 selitetään tarkemmin käyttöliittymäkuvassa näkyvien elementtien tarkoituksia ja toimintaa.

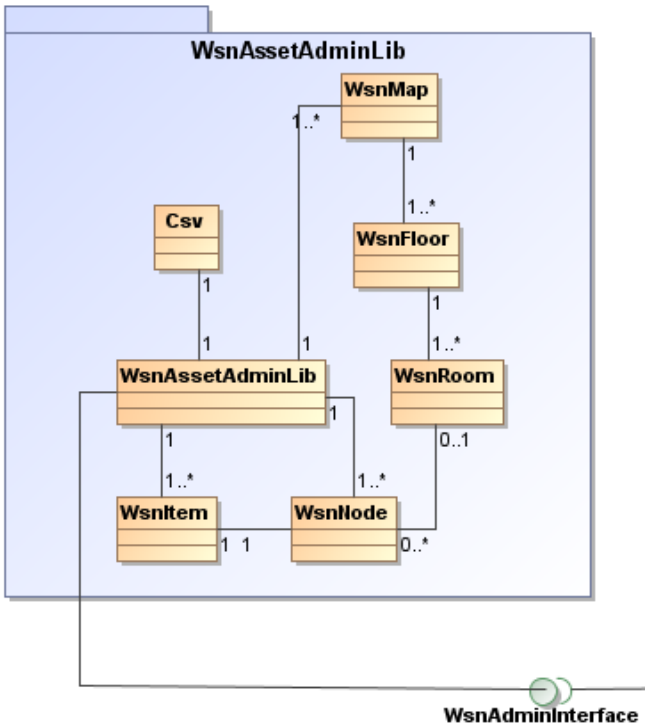


Kuva 8: Pääkäyttäjän käyttöliittymänäkymä.

4.1 Pääkäyttäjän sovelluksen moottori

Pääkäyttäjän sovelluksen ytimenä toimii sen logiikkamoottori. Se on toteutettu dynaamisesti linkitettävänä kirjastona ja se tarjoaa rajapintafunktiot sovellukseen liitettävälle käyttöliittymäkomponentille. Moottorin ensisijainen tehtävä on käsitellä useita erimuotoisia csv-tiedostoja. Tiedostoista luetaan tiedot karttapohjista, alueista, huoneista, mittalaitteista ja esineistä. Luetut tiedot tallennetaan moottorin tietorakenteisiin. Moottori hoitaa myös sovellukseen lisättyjen ja / tai muokattujen tietojen tallentamisen csv-tiedostoihin. Moottorin tarjoaman rajapinnan (*WsnAdminInterface*) kautta hoidetaan lisäksi kyselyt esineiden ja alueiden sijainneista.

Oliot: *Csv*, *WsnAssetAdminLib*, *WsnFloor*, *WsnItem*, *WsnMap*, *WsnNode* ja *WsnRoom* muodostavat pääkäyttäjän sovelluksen moottorin. Suurin osa logiikasta käsitellään *WsnAssetAdminLib*-luokan funktioilla. Se toteuttaa *WsnAdminInterface*-rajapinnassa esitellyt funktiot. *WsnAdminInterface*-rajapinnan funktiot ja niiden toiminnan kuvaus löytyy liitteestä 1. Kuva 9 havainnollistaa pääkäyttäjän moottorin olioiden suhteita toisiinsa.



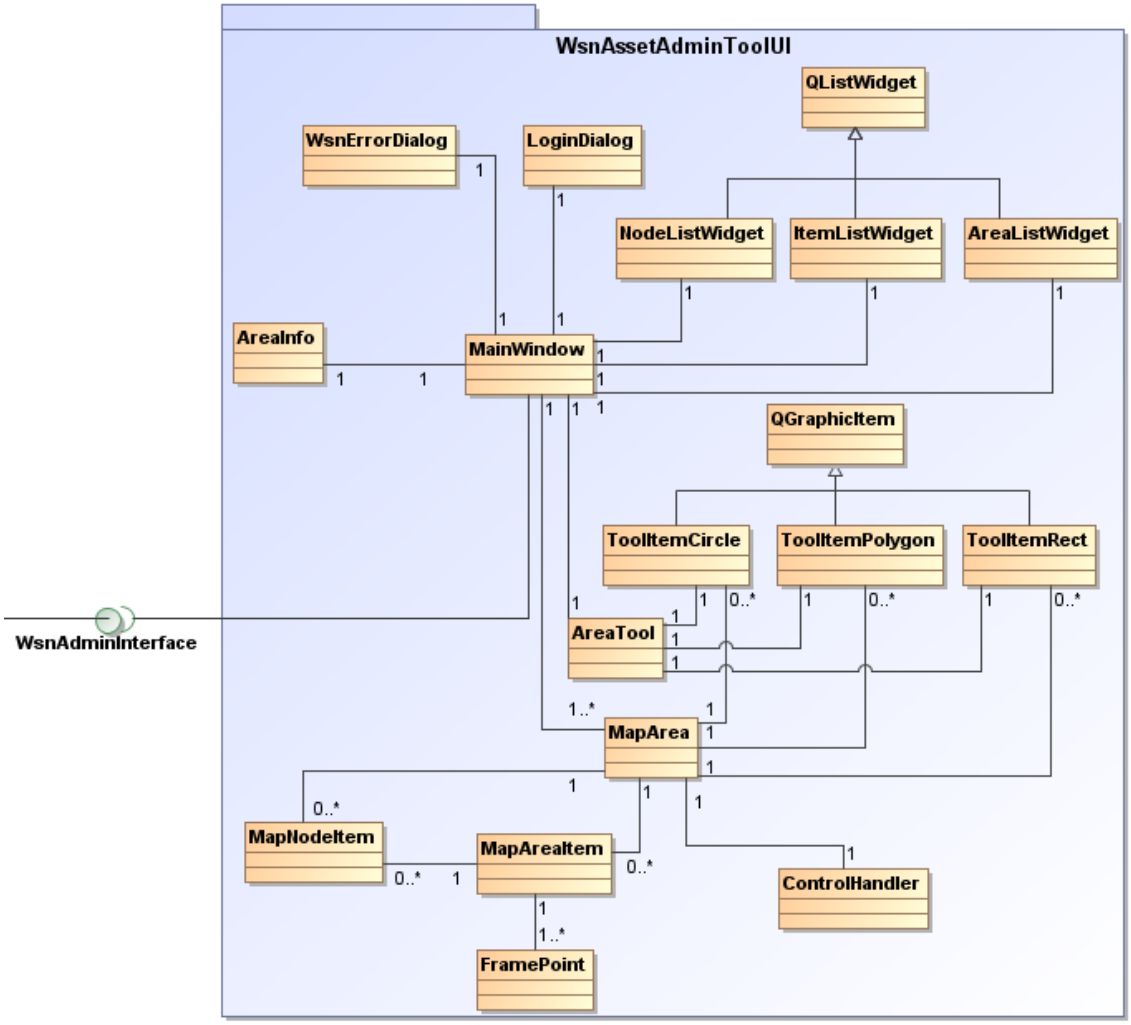
Kuva 9: Pääkäyttäjän sovelluksen logiikkamoottorin kuvaus luokkakaaviona.

WsnAssetAdminLib-luokka käsittelee kaikki suurimmat tehtävät, kuten karttapohjien, alue-, mittalaite-, ja esinetietojen lataamisen ja tallentamisen. Sen kautta kulkevat myös käyttöliittymän pyynnöt, joilla haetaan ja asetetaan tieto esimerkiksi mittalaitteen sijainnista karttapohjalla ja mihin kerrokseseen / rakennukseseen mittalaite juuri asetettiin. *WsnMap*-luokka sisältää tiedon rakennuksen nimestä ja *WsnFloor* sisältää tiedon kerroksen nimestä, sekä hakemistopolusta ja tiedoston nimestä, joiden takaa löytyy kyseisen kerroksen karttapohjan kuvatiedosto.

WsnItem, *WsnNode* ja *WsnRoom* sisältävät lähinnä tietoa, missä rakennuksessa tai kerroksessa joku esine, mittalaite tai huone sijaitsee. Huoneeseen liitetään tietoa muun muassa sen leveydestä, pituudesta ja kierrosta karttapohjan päällä. Mittalaite sisältää tiedon esimerkiksi laitteen tyypistä ja sijainnista.

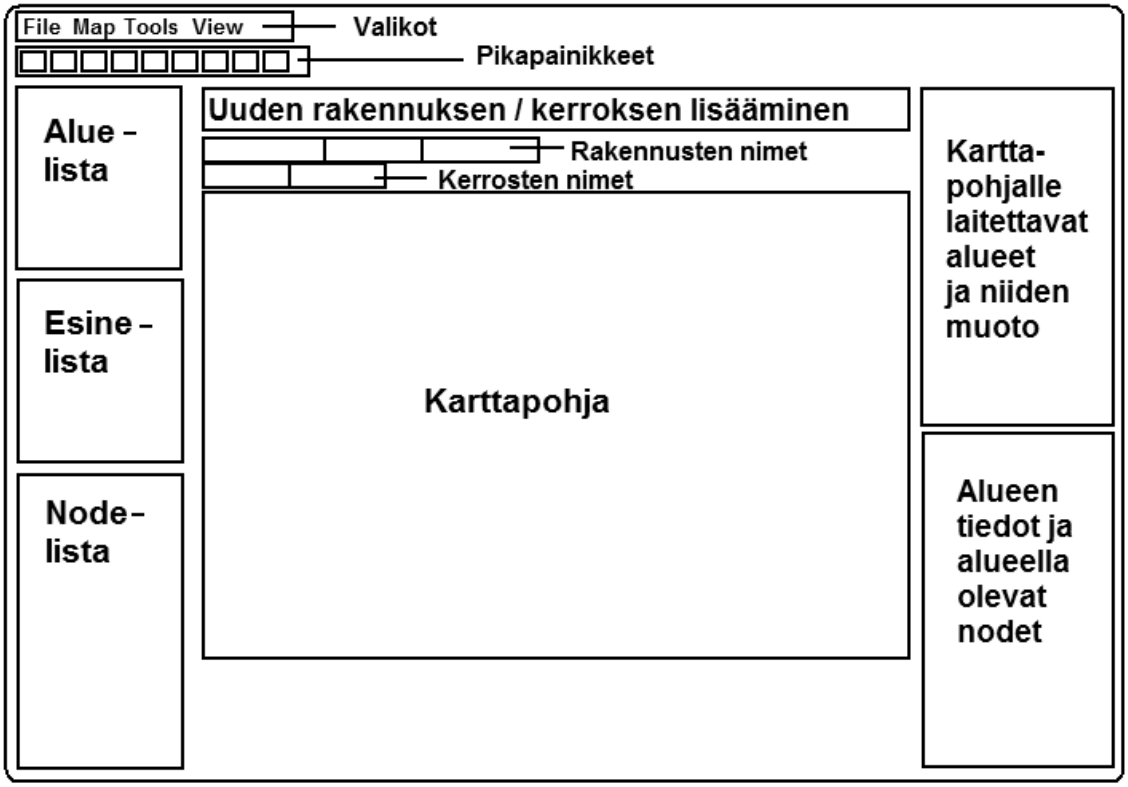
4.2 Pääkäyttäjän sovelluksen käyttöliittymä

Käyttöliittymän tärkein olio on *MainWindow*, joka sisältää ohjelman päänäkymän ja kaikki siihen kuuluvat valikot, painikkeet ja säätimet. *MainWindow*-luokan alussa kysytään käyttäjän tunnus ja salasana, ennen kuin sovelluksen muut tiedot ladataan. Kirjautumisen jälkeen käyttöliittymään ladataan viimeksi muokatun projektin tiedot ja tämän jälkeen alueiden ja mittalaitteiden sijainnit asetetaan projektin karttapohjille. Kuva 10 havainnollistaa käyttöliittymän olioiden keskinäisiä suhteita.



Kuva 10: Pääkäyttäjän sovelluksen käyttöliittymän kuvaus luokkakaaviona.

Kuvassa 11 on esitetty päänäkymän käyttöliittymäelementtien sijoittelu ja alueiden jaottelu. Osa käyttöliittymän elementeistä vastaa suoraan käyttöliittymän olioita (kuva 10). Esimerkiksi *Area-*, *Item-* ja *NodeListWidget*-oliot sisältävät kaikki toiminnot, jotka ovat sisällytetty alue-, esine- ja mittalaitelistoihin.



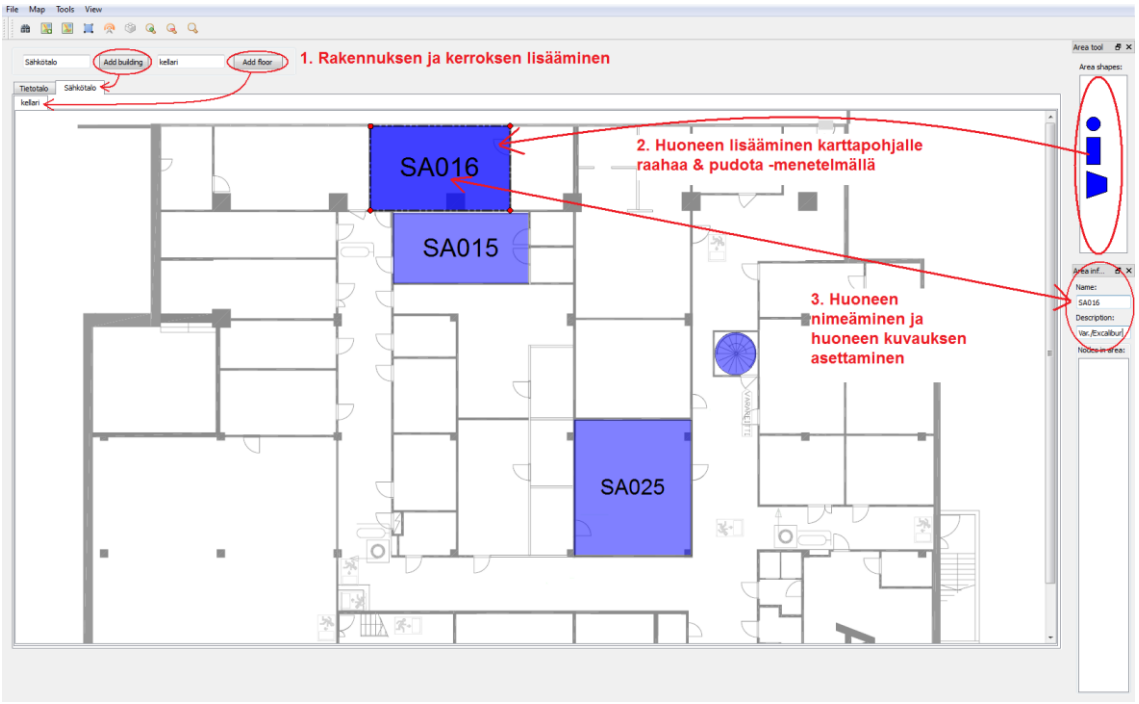
Kuva 11: Pääkäyttäjän näkymän käyttöliittymäelementtien sijoittelu.

Karttapohja on päänäkymän suurin ja tärkein elementti. Siihen ladataan kuvatiedostosta karttapohja, jonka päälle asetetaan huoneet näkymän oikeassa reunassa olevasta alueiden muotoja sisältävästä grafiikkanäkymästä. Se sisältää 3 erityyppistä huoneen muotoa (suorakulmio, ellipsi ja monikulmio), joista raahaa & pudota -menetelmän avulla alue lisätään karttapohjalle. Tämän jälkeen alueen kokoa voi muuttaa vetämällä alueen päälle ilmestyvistä kulmapisteistä, tai aluetta voidaan kiertää pyörittämällä hiiren rullaa. Alueelle voi antaa nimen ja kuvauksen syöttämällä pääkäyttäjän näkymän oikeassa alakulmassa olevan elementin tekstikenttiin tekstiä. Alueen nimen voi vaihtoehtoisesti klikata suoraan vasemmassa yläkulmassa olevasta *Aluelistasta*.

Mittalaitteet lisätään kartalle näkymän vasemmassa alakulmassa olevasta *Nodelistasta* raahaa & pudota -menetelmällä. Jos mittalaitetta esittävä ympyrä leikkaa jotain kartalle lisättyä aluetta, se lisätään alueen mittalaitelistalle. Poistamalla alue tai mittalaite karttapohjalta alueen nimi tai mittalaitteen tunnistenumero siirtyy takaisin *Mittalaite- / Aluelistalle*. Kun karttapohjalla on joku mittalaite valittuna ja vasemmalta napsautetaan *Esinelistalta* jotain esinettä, siirtyy tieto esineestä myös mittalaitteen tietoihin ja myöhemmin myös pääkäyttäjän moottorin tietorakenteisiin lisätään tieto mittalaitteen ja esineen yhteenliitoksesta.

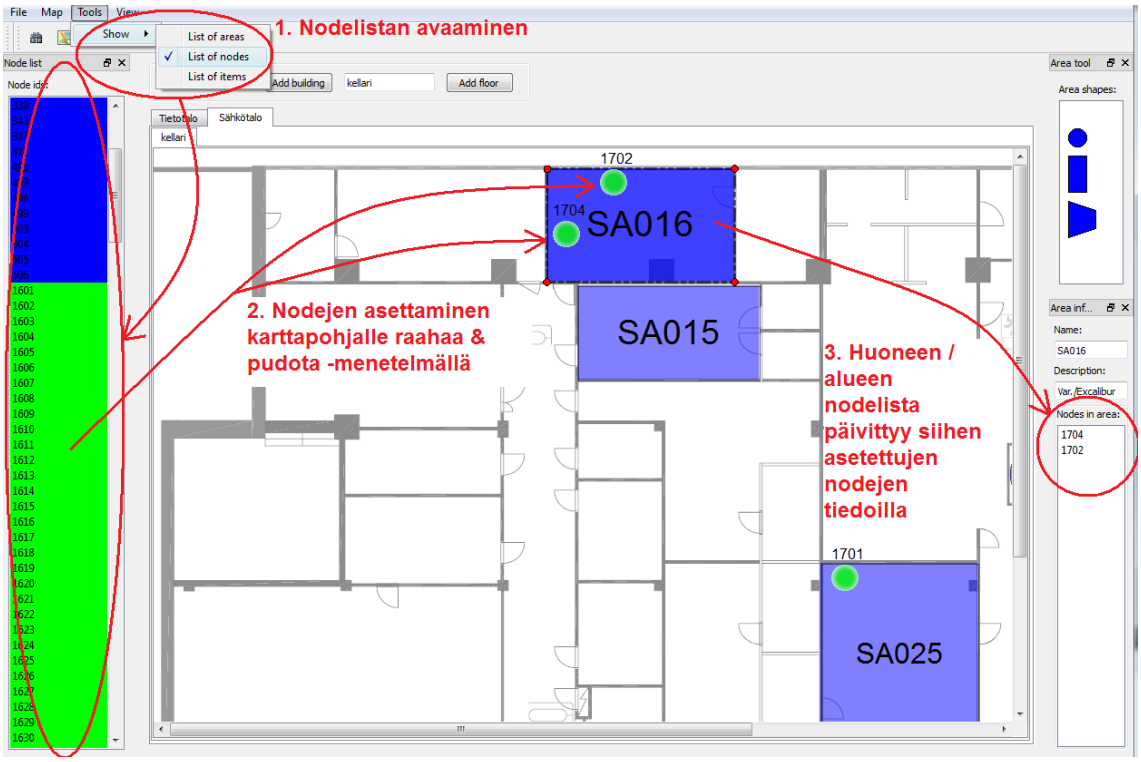
4.3 Esimerkkejä käyttöliittymästä

Seuraavassa kuvasarjassa on esitetty tärkeimmät pääkäyttäjän sovelluksen avulla suoritettavat toiminnot. Kuvassa 12 on esitetty rakennuksen ja kerroksen lisääminen (kohta 1), huoneen lisääminen karttapohjalle (kohta 2) ja huoneen tietojen lisääminen pääkäyttäjän käyttöliittymän avulla.



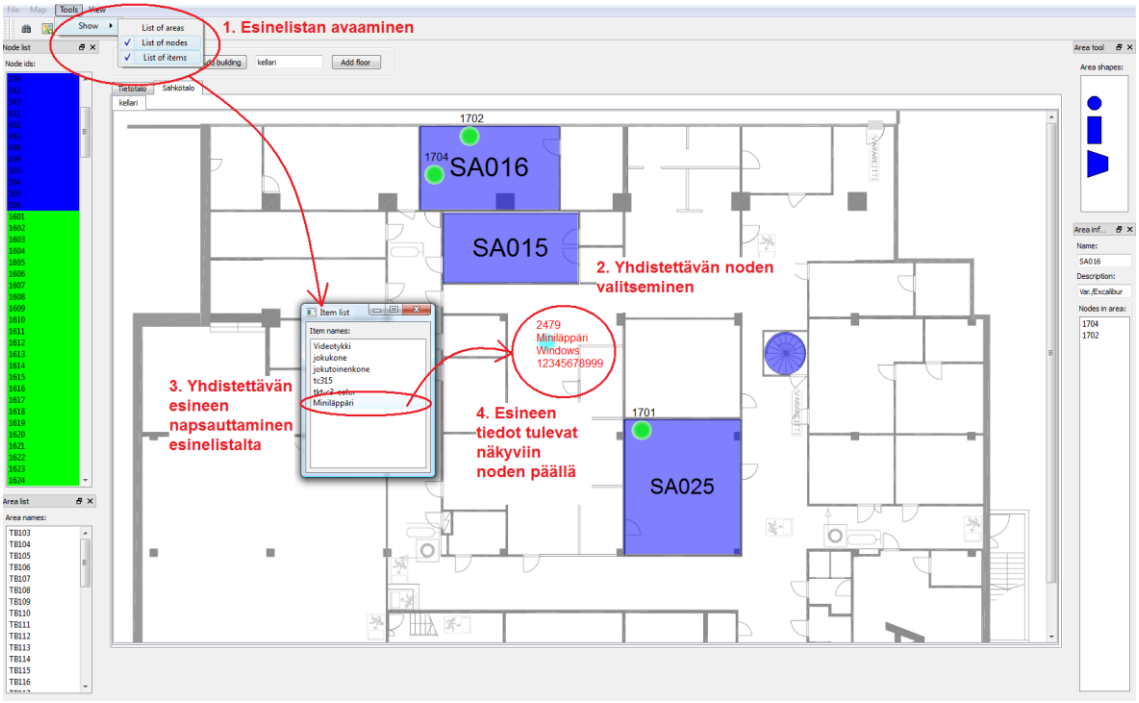
Kuva 12: 1. Rakennuksen ja kerroksen lisääminen käyttöliittymässä. 2. Huoneen lisääminen karttapohjan päälle. 3. Huoneen nimeäminen ja huoneen kuvauksen asettaminen.

Kuvassa 13 on esitetty mittalaitteen asettaminen huoneen sisään pääkäyttäjän sovelluksen käyttöliittymän avulla. Ensin avataan mittalaitelistaus päävalikon kautta: *Tools->Show->List of nodes*. Seuraavaksi listalta valitaan halutun mittalaitteen tunnistenumero, joka siirretään karttapohjalle raahaa & pudota -toimintoa käyttämällä. Mittalaitetta esittävän ympyrän leikatessa jotain huonetta se lisätään kyseisen huoneen mittalaitelistaan. Mittalaitelistalla vihreällä pohjalla merkityt mittalaitteet esittävät reitittäviä mittalaitteita, tummansiniset esittävät yhdyskäytävälaitteita ja vaaleansiniset esittävät liikkuvia mittalaitteita.



Kuva 13: 1. Mittalaitelista avataan valikosta. 2. Mittalaitteet siirretään listasta karttapohjalle raahaa & pudota -toiminnon avulla. 3. Huoneen mittalaitelistaan lisätään automaattisesti siihen asetettujen mittalaitteiden tunnistenumerot.

Kuvassa 14 tarkastellaan mittalaitteen yhdistämistä esineen kanssa. Ensin avataan esinelistaus päävalikon kautta: *Tools->Show->List of items*. Tämän jälkeen napsautetaan se mittalaite aktiiviseksi, johon esine halutaan yhdistää. Käyttöliittymä ilmoittaa mittalaitteen olevan valittuna välkyttämällä sitä. Seuraavaksi napsautetaan esinelistalta esinettä, joka halutaan yhdistää valittuna olevan mittalaitteen kanssa. Lopuksi käyttöliittymässä annetaan palaute valmiista esineen ja mittalaitteen liitoksesta lisäämällä mittalaitteen päälle esineen nimi. Jos sama mittalaite valitaan uudestaan, näytetään siihen yhdistetyn esineen nimi, tyyppi ja laiterekisterinumero punaisella tekstillä, joka ilmaisee käyttäjälle, että mittalaite on valittuna.



Kuva 14: 1. Esinelistauksen avaaminen. 2. Yhdistettävänä olevan mittalaitteen valinta. 3. Yhdistettävänä olevan esineen napsauttaminen esinelistalta. 4. Esineen ja mittalaitteen yhdistäminen on valmis.

4.4 Käyttöliittymän ja moottorin toteutuksen ongelmia ja ratkaisuja

Seuraavassa on esitelty neljä pääkäyttäjän sovelluksen toteutuksen vaikeinta kohtaa. Muitakin vaikeita osuuksia työssä oli, mutta seuraavien kohtien avaaminen saattaa auttaa henkilöitä, jotka etsivät ratkaisua esimerkiksi graafisten objektien käsittelyyn Qt-ympäristössä. Seuraaviin kohtiin ei löytynyt täysin valmiita ratkaisuja vuoden 2011 lokakuuhun mennessä, siksi ne on nostettu esille. Ainoastaan dynaamisen kirjaston lataaminen löytyi Qt:n dokumentaatiosta [29] osittain, joka ei aivan suoraan projektiin siirrettynä toiminut. Tämä voi olla korjattu, koska Qt:sta ilmestyi uudempi versio työtä tehdessä.

4.4.1 Dynaamisen kirjaston lataaminen käyttöliittymäsovelluksessa

Qt:lla dynaamista kirjastoa luodessa täytyy ensin projektin projekti-tiedostoon (pro -tiedosto) lisätä seuraava rivi:

CONFIG += qt plugin

Seuraavaksi voidaan luoda kirjaston rajapintatiedosto, esimerkiksi *wsninterface.h*. Yksinkertaisimmillaan tiedosto sisältää eri olioiden esiesittelyjä ja virtuaalifunktioita [32]. Seuraavassa on esitetty esimerkki rajapintatiedostosta:

```
#ifndef WSNINTERFACE_H
#define WSNINTERFACE_H

#include <QtGlobal>
class QString;


class WsnInterface {
public:
  virtual QString getErrorMessage() = 0;
        ...
        ...
};


Q_DECLARE_INTERFACE(WsnInterface, "WsnLib/1.0")
#endif // WSNINTERFACE_H
```

Rajapinnan toteuttavan luokan otsikkotiedostossa (h-tiedosto) tulee mainita makro Q_INTERFACES([Halutun rajapinnan nimi]) ja otsikkotiedoston toteuttavan tiedoston (cpp-tiedosto) loppuun lisätään rivi:

```
Q_EXPORT_PLUGIN2( [Pluginin nimi], [Luokan nimi])
```

Seuraavassa on esimerkki yksinkertaisen otsikkotiedoston sisällöstä (wsnlib.h):

```
#ifndef WSNLIB_H
#define WSNLIB_H
#include <QObject>
class WSNLIBSHARED_EXPORT WsnLib : public QObject, WsnInterface {
    Q_OBJECT
    Q_INTERFACES(WsnInterface)
public:
    WsnLib();
    QString getErrorMessage();
        ...
        ...
private:
      ...
      ...
};

#endif // WSNASSETADMINLIB_H
```

Seuraavassa on esimerkki otsikkotiedoston toteuttavasta tiedostosta (wsnlib.cpp):

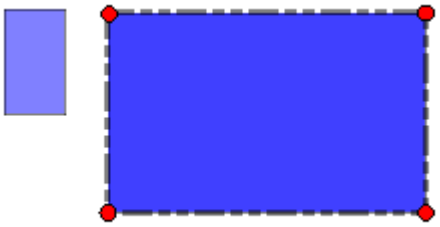
```
#include "wsnlib.h"

QString WsnLib::getErrorMessage() {
    return QString("Error");
}

Q_EXPORT_PLUGIN2(wsnlib, WsnLib)
```

Dynaaminen kirjasto ladataan käyttöliittymäprojektissa seuraavien kohtien mukaisessa järjestyksessä:

1. Lisätään #include-esikääntäjäkomento [32] projektin tiedostoon, jolla määritetään, mistä rajapinnan esittelytiedosto löytyy. Tämä voidaan tehdä esimerkiksi mainwindow.h -tiedostoon:
   ```
   #include "../../WsnLib/WsnLib/wsninterface.h"
   ```
2. Lisätään #include-esikääntäjäkomento, jolla kerrotaan, että projektissa käytetään QPluginLoader-luokan funktioita:
   ```
   #include <QPluginLoader>
   ```
3. Ladataan kirjaston rajapinta. Esimerkki windows-koneelle lataamisesta:
   ```
   // .dll –tiedosto löytyy hakemistosta ../plugins/
   QPluginLoader pluginLoader("plugins/WsnLib.dll");
       QObject* pObject = pluginLoader.instance();

   // Jos hakemisto ja tiedoston nimi oli oikein, niin...
   if(pObject)
       {
           qDebug() << "Plugin loaded successfully!!!";
           pWsnInterface_ = qobject_cast<WsnInterface*> (pObject);

   // pWsnInterface_ on mainwindow.h luokan privaattimuuttuja:
   // WsnInterface* pWsnInterface_;
     ...
     ...
   ```

4.4.2 Graafisen objektin koon ja muodon muuttaminen dynaamisesti

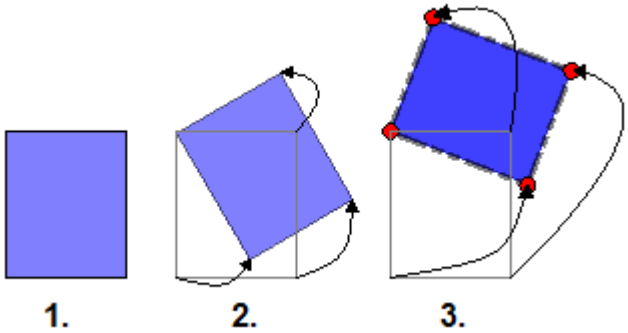
Graafiselle objektille [33] luodaan kehys, joka koostuu graafista objektia rajaavan suorakulmion (bounding rect) kulmapisteistä. Kulmapisteistä vetämällä graafisen objektin muoto ja / tai koko muuttuu (riippuen siitä onko kyseessä suorakulomio, ellipsi vai monikulmion muotoinen objekti). Graafiset objektit, joiden koko ja muoto on muunneltavissa kuvastavat pääkäyttäjän sovelluksessa huoneita, käytäviä ja muita samankaltaisia alueita. Kuvat 15 ja 16 havainnollistavat graafisen objektin ja siihen liitetyn kehyksen toimintaa. Kun pääkäyttäjän sovellukseen karttapohjalle asetettua aluetta napsautetaan, muuttuu se valituksi ja sen ympärille piirretään kehys. Kehykseen kuuluu katkoviivoilla rajattu polku ja kehyksen kulmissa olevat punaiset kulmapisteet.



Kuva 15: Vasemmalla alkuperäinen graafinen objekti ja oikealla kehyksellä varustettu graafinen objekti, joka on muodostettu alkuperäisestä objektista vetämällä kulmapisteestä.

4.4.3 Graafisen objektin kiertäminen

Graafisen objektin kiertämisen yhteydessä täytyy kehyksen kulmapisteille laskea uudet sijainnit (kuva 16).



Kuva 16: Graafisen objektin kiertäminen vaiheittain. Kohdassa 1 on alkuperäinen objekti, kohdassa 2 on välivaihe ja kohdassa 3 on objektin lopullinen kiertymä kehyksen kanssa. Objektin kierto tapahtuu sovelluksessa hiiren rullaa pyörittämällä, kun hiiren kursori on graafisen objektin päällä.

2D-grafiikkaa piirrettäessä objektin kiertämisen kaava on esitetty seuraavassa [34, 35]:

q = alkuperäinen kulma, f = kääntökulma.

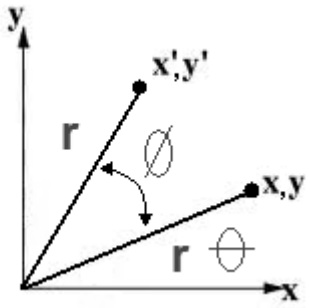
$$x = r \cos q \quad (1)$$
$$y = r \sin q \quad (2)$$

$$x' = r \cos (q + f) = r \cos q \cos f - r \sin q \sin f \quad (3)$$
$$y' = r \sin (q + w) = r \sin q \cos f + r \cos q \sin f \quad (4)$$

Siten:

$$x' = x \cos f - y \sin f \quad (5)$$
$$y' = y \cos f + x \sin f \quad (6)$$

Vanhat koordinaatit ovat (x, y) ja uudet koordinaatit ovat (x', y'). Kuva 17 havainnollistaa, kuinka koordinaatit muuttuvat kiertämisen yhteydessä.



Kuva 17: Esimerkki graafisen objektin kiertämisestä kaksiulotteisessa koordinaatistossa [35].

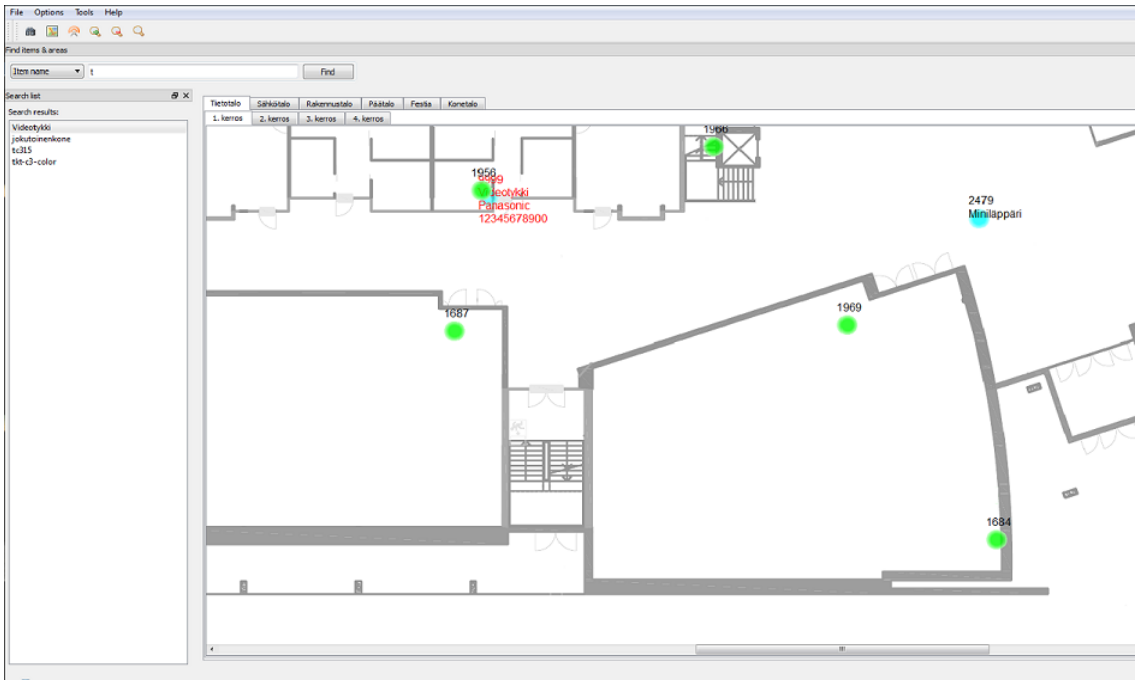
4.4.4 Graafisen objektin kopioiminen ja liittäminen

Graafisen objektin muodon, koon ja kiertymän kopioinnin tarkoituksena on nopeuttaa karttaeditorissa usein suoritettavia toimintoja. Kopioidun objektin liittäminen hiiren kursorin osoittamaan kohtaan nopeuttaa karttaeditorissa suoritettavaa toimintoa, jossa useita uusia samanmuotoisia alueita luodaan karttapohjalle. Toteutus on tehty ottamalla valittuna olevan alueen muoto-, koko- ja rotaatiotiedot väliaikaiseen muuttujaan (osoitin *MapAreaItem* -tyyppiseen olioon). Alla olevassa ohjelmakoodissa on esitetty ratkaisuesimerkki:

```
// Liittää kopioidun alueen parametrina olevaan kohtaan karttapohjalla
void MapArea::pasteMapAreaItem(QPoint pos) {
  ...
  ...
  // Luodaan uusi Alue, jolle annetaan kopioitavan alueen muoto ja
  // asetetaan sen Z-arvo (onko objekti karttapohja-, alue- vai
  // mittalaitekerrokseen kuuluva graafinen objekti)
  MapAreaItem* item = new MapAreaItem(pCopyMapAreaItem_->getShape());
  item->setZValue(AREALAYER);

  // Kuvataan objektin sijainti vastaamaan hiiren kursorin sijaintia
  QPoint globalPoint = this->mapFromGlobal((pos));
  QPointF scenePoint = mapToScene(globalPoint);

  // Asetetaan kopioidun objektin leveys, korkeus ja kiertymä
  // liitettävälle objektille
  item->setWidth(pCopyMapAreaItem_->boundingRect().width());
  item->setHeight(pCopyMapAreaItem_->boundingRect().height());
  item->setRotation(pCopyMapAreaItem_->rotation());

  // Asetetaan objektin sijainti grafiikka-alueella (karttapohja),
  // annetaan sille sijainti, asetetaan objekti valituksi, lisätään se
  // grafiikka-alueelle ja liitetään sille kehys
  item->setPos(scenePoint);
  item->setAreaItemSelect(true);
  pScene_->addItem(item);
  addFrameToRect(item);
```

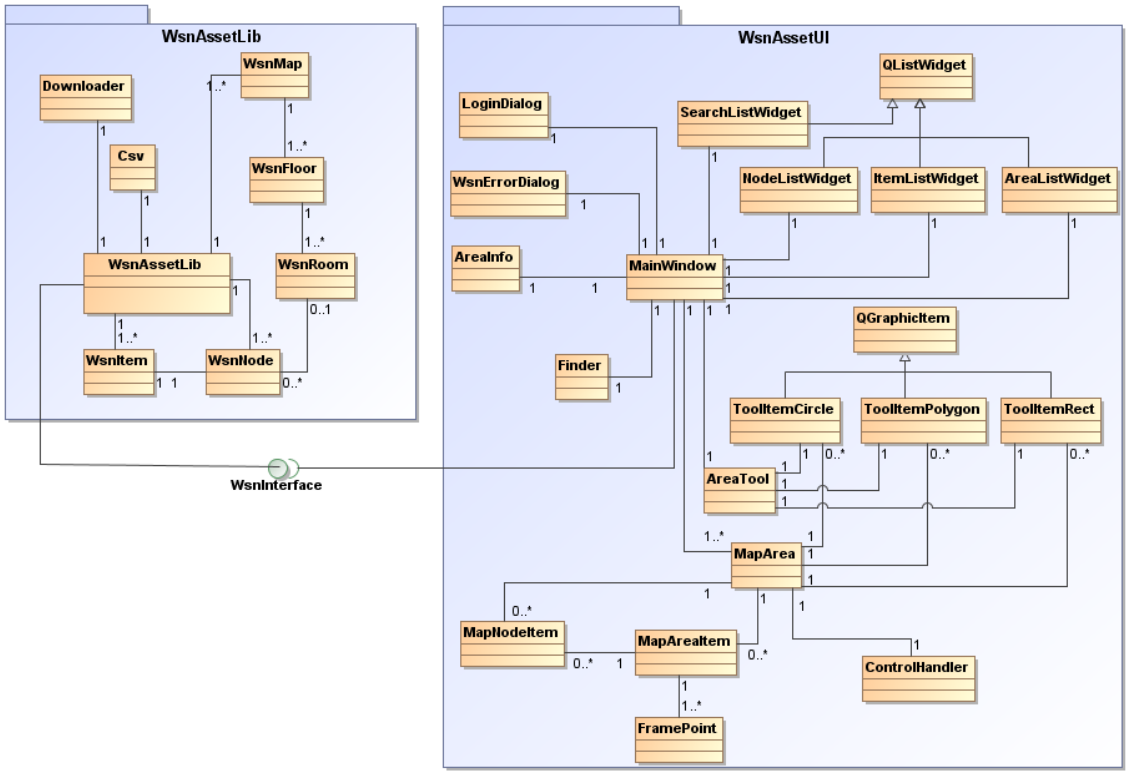
5 PERUSKÄYTTÄJÄN SOVELLUS

Peruskäyttäjän sovelluksen tarkoituksena on esittää paikannettavan omaisuuden sijainti käyttäjälle helposti ymmärrettävässä muodossa. Sijainnin lisäksi peruskäyttäjälle näytetään muita tärkeitä tietoja haetuista alueista ja esineistä. Käyttäjän hakema tieto voi olla esimerkiksi esineen tunnetut sijainnit kuluneen viikon ajalta. Kuvassa 18 on kuvankaappaus peruskäyttäjän käyttöliittymänäkymästä. Kohdassa 5.1 on kuvattu peruskäyttäjän moottoriin kuuluvia olioita ja niiden toimintoja. Peruskäyttäjän käyttöliittymän toimintoja ja käyttöliittymäelementtien tarkoituksia on selitetty tarkemmin kohdissa 5.2 ja 5.3.



Kuva 18: Peruskäyttäjän käyttöliittymänäkymä.

Peruskäyttäjän sovellus koostuu pääkäyttäjän sovelluksen tapaan logiikkamoottorista ja sen tarjoamasta rajapinnasta, sekä käyttöliittymästä (kuva 19). Peruskäyttäjän sovelluksessa on samat luokat, kuin pääkäyttäjän sovelluksessa, mutta näiden lisäksi *WsnAssetLib*-kirjasto sisältää luokan *Downloader* ja käyttöliittymäkomponentti (*WsnAssetUI*) sisältää kaksi uutta luokkaa: *Finder* ja *SearchListWidget*.



Kuva 19: Peruskäyttäjän sovellukseen kuuluvat komponentit ja luokat.

5.1 Peruskäyttäjän sovelluksen moottori

Peruskäyttäjän sovelluksen moottori eroaa pääkäyttäjän sovelluksen moottorista siten, että *WsnAssetAdminLib*-luokka on korvattu *WsnAssetLib*-luokalla, sekä *Downloader*-luokan lisäämisellä. Moottorin pääluokka *WsnAssetLib* käsittelee tietoa rakennuksista, kerroksista, huoneista, mittalaitteista ja esineistä siten, että peruskäyttäjä ei voi muutella esimerkiksi huoneiden sijaintitietoja tai liikkuvaan mittalaitteeseen liitetyn esineen tietoja. Peruskäyttäjän sovellus on tarkoitettu esineen viimeaikaisten sijaintien raporttien pyytämiseen ja mahdollisten hälytysten lähettämiseen, joten on hyödyllistä että peruskäyttäjä ei pääse sotkemaan vahingossa karttapohjille sijoitettujen huoneiden tai kiinteiden mittalaitteiden sijainteja.

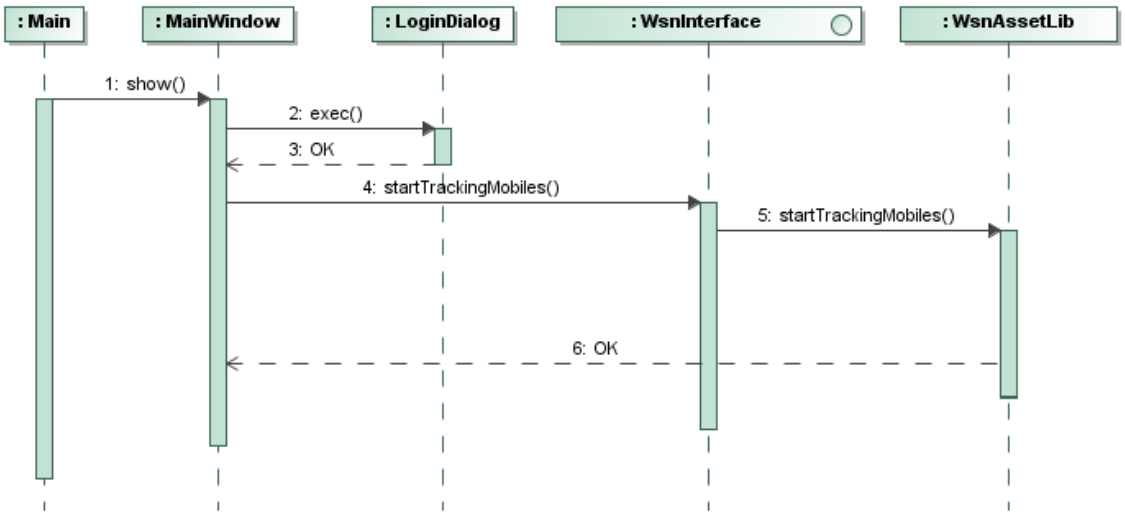
Peruskäyttäjän sovelluksen moottori lukee pääkäyttäjän sovelluksella muokattuja tiedostoja joihin on tallennettu eri karttapohjat, huoneiden ja mittalaitteiden koordinaatit, sekä liikkuviin mittalaitteisiin yhdistettyjen esineiden tunnistenumerot. *WsnAssetLib*-luokka sisältää suodatuksen hakusanaa vastaan, jolla voidaan etsiä huonetta, mittalaitetta tai esinettä. Huonetta voidaan etsiä huoneen tunnisteella ja huoneen kuvauksen avulla. Mittalaite löydetään tunnistenumeron avulla ja esineitä voidaan hakea laiterekisterinumeron tai esineen kuvauksen avulla.

Downloader-luokka lataa http-protokollan avulla TUTWSN-verkon keräämää dataa liikkuvien mittalaitteiden naapurilaitetiedoista. TUTWSN-verkon mittalaitteet lähettävät tiedon naapurilaitteistaan viiden minuutin välein, mutta tiedot lähetetään eri ajanhetkinä yhdyskäytävälaitteiden (gatewaynode) kautta palvelimelle ja tietokantaan. Tämän vuoksi *WsnAssetLib*-luokassa laukaistaan tapahtuma, joka lataa mittalaitteiden viimeisimmät sijainnit kahden ja puolen minuutin välein. *Downloader*-luokka sisältää myös toiminnon, jonka avulla ladataan sensoriverkon tapahtumat viimeksi kuluneen kuukauden ajalta. Suuren datamäärän vuoksi ladattavat tiedot sisältävät vain liikkuvien mittalaitteiden lähettämiä naapuritietoja, joista liikkuvien mittalaitteiden sijainnit voidaan selvittää.

5.2 Peruskäyttäjän sovelluksen käyttöliittymä

Peruskäyttäjän käyttöliittymän tärkein luokka on *MainWindow*, kuten pääkäyttäjän käyttöliittymässä. Se lataa ensimmäisenä instanssin peruskäyttäjän moottorista (*WsnAssetLib.dll*), jonka jälkeen *WsnInterface*-rajapinnan funktioita voidaan kutsua (liite 2). Tämän jälkeen avataan sisäänkirjaantumisen ponnahdusnäkymä, jonka jälkeen ladataan eri rakennusten ja niiden jokaisen kerroksen karttapohjat ja graafiset elementit karttapohjien päälle. Käyttöliittymästä kutsutaan myös *WsnInterface*-rajapinnan *startTrackingMobiles()*-funktiota, joka käynnistää tapahtuman, jolla haetaan liikkuvien mittalaitteiden viimeksi tiedetyt sijainnit.

Kuva 20 esittää peruskäyttäjän sovelluksen käynnistysvaihetta ja liikkuvien mittalaitteiden seurannan alkamista. Aluksi *main*-luokassa luodaan *MainWindow*-tyyppinen olio, jossa käydään tutkimassa *LoginDialog*-olion avulla, onko käyttäjänimi ja salasana oikein. Tämän jälkeen luodaan yhteys *WsnInterface*-rajapinnan avulla *WsnAssetLib*-kirjastoon, josta kutsutaan liikkuvien mittalaitteiden paikannuksen aloittavaa funktiota.

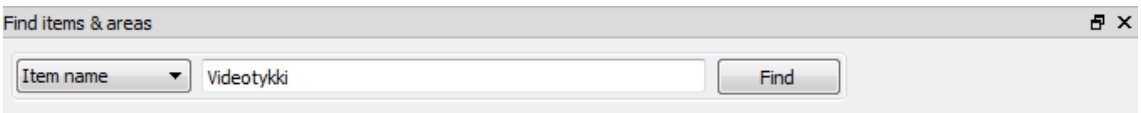


Kuva 20: Liikkuvien mittalaitteiden seurannan käynnistymistä kuvaava sekvenssikaavio.

Peruskäyttäjän käyttöliittymän arkkitehtuurin rakenne on samankaltainen, kuin pääkäyttäjän sovelluksessa. Uusina luokkina peruskäyttäjän käyttöliittymässä ovat *Finder* ja *SearchListWidget*. *Finder*-luokka kuvastaa käyttöliittymän tekstikenttää, vetovalikkoa ja painiketta, joiden avulla etsitään haluttua esinettä, huonetta tai mittalaitetta. Alasvetovalikossa on 5 valmiiksi annettua vaihtoehtoa, joilla edellä mainittuja asioita voidaan etsiä:

- alueen (huoneen) tunnisteen,
- alueen (huoneen) kuvauksen,
- esineen nimen,
- esineen laiterekisterinumeron, tai
- mittalaitteen tunnistenumeron mukaan.

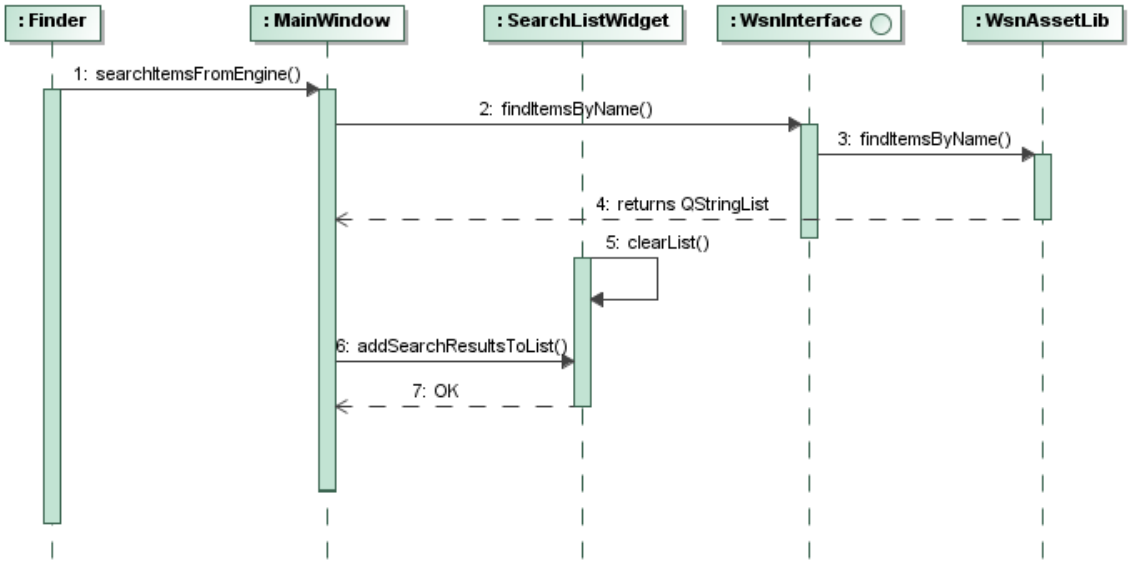
Alasvetovalikon oikealla puolella olevaan tekstikenttään syötetään hakusana tai osa numerosarjasta, joka liittyy esineen laiterekisterinumeroon tai mittalaitteen tunnisteeseen. Kun tekstikenttään on syötetty haluttu merkkijono, painetaan sen oikealla puolella olevaa ”**Find**”-painiketta. Tämän jälkeen käyttöliittymään avautuu lista vaihtoehdoista, jotka kyseisillä hakuehdoilla löytyi. Kuvassa 21 näkyvät *Finder*-luokkaan kuuluvat elementit.



Kuva 21: Finder-luokan elementit. Vasemmalla alasvetovalikko, keskellä tekstikenttä, johon hakusana syötetään ja oikealla painike, joka käynnistää haun.

Lista johon hakuehtojen tulokset on haettu, on toteutettu *SearchListWidget*-luokan avulla. Listaan haettua vaihtoehtoa napsauttamalla käyttöliittymä lähettää signaalin *MainWindow*-luokalle, joka edelleen etsii *WsnInterface*:n rajapintafunktioiden avulla, missä rakennuksessa ja monennessako kerroksessa haettu esine, huone tai mittalaite on. Tämän jälkeen kutsutaan kohteeseen liittyvän *MapArea*-luokan *centerOnAreaByName(Qstring areaName)*-funktiota, joka keskittää näkymän haettuun esineeseen / huoneeseen / mittalaitteeseen.

Kuvassa 22 on sekvenssikaavio hakuoperaation suorittamisesta peruskäyttäjän sovelluksessa. Esinettä etsitään siinä nimen perusteella. Aluksi *Mainwindow*-luokasta lähetetään etsintäpyyntö peruskäyttäjän moottorille (*WsnAssetLib.dll*) *WsnInterfacen* kautta. Jos hakusanalla löytyy esineitä, moottori palauttaa *QstringList*-olion [36] löydetyistä esineistä. Tämän jälkeen *SearchListWidget* tyhjennetään vanhoista tuloksista ja sitten sinne lisätään hakusanalla löytyneet esineet. Löydettyjä tuloksia napsauttamalla valitaan välilehdiltä oikea rakennus ja kerros, sekä keskitetään grafiikkanäkymä haluttuun esineeseen.

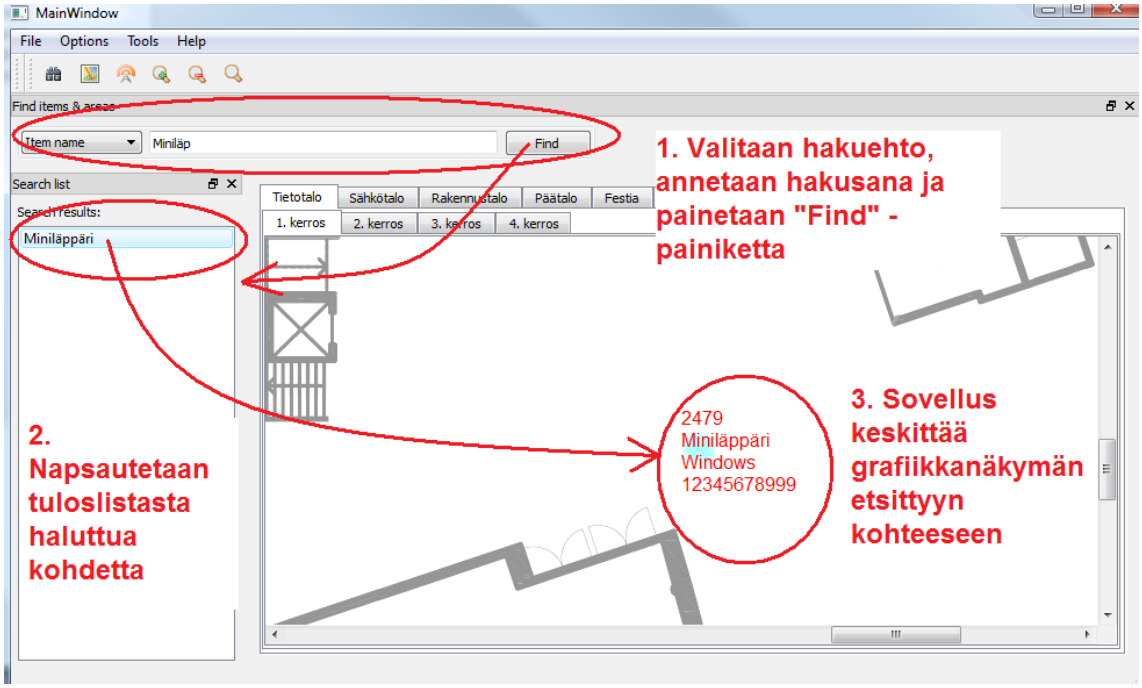


Kuva 22: Sekvenssikaavio sovelluksen hakuoperaatiosta, jossa etsitään esinettä nimen mukaan.

5.3 Käyttöliittymäesimerkkejä

Kuvat 23 ja 24 esittävät yleisimpiä peruskäyttäjän sovelluksella suoritettavia toimintoja. Kuvassa 23 on esimerkki esineen etsimisestä peruskäyttäjän käyttölittymän avulla. Ensin valitaan alasvetovalikosta (*Finder*-luokka, kuva 21) hakuehdoksi esineen nimi (*Item name*), kirjoitetaan tekstikenttään hakusana ja painetaan ”**Find**”-painiketta. Tämän jälkeen karttapohjan vasemmalle puolelle aukeaa tuloslistaus hakusanaa vastaavista esineistä (kohta 2). Kun tuloslistalta painetaan haluttua kohdetta, käyttölittymään

asetetaan aktiiviseksi se rakennus ja kerros, jossa esine sijaitsee. Lisäksi grafiikkanäkymä keskitetään esineeseen ja se asetetaan valituksi, jolloin käyttäjän huomio keskittyy kohteeseen (kohta 3).



Kuva 23: Halutun kohteen etsintä peruskäyttäjän sovelluksen avulla välivaiheineen.

Kuvassa 24 on esimerkki, kuinka kerroksen kaikki mittalaitteet saadaan näkyviin karttapohjan päällä samanaikaisesti, valitsemalla päävalikosta: *Options->Show all nodes*. *Options*-valikon alta löytyy myös asetukset, joilla saadaan kerroksen kaikki huoneet tai kaikki kerroksessa olevat liikkuvat mittalaitteet näkyville. Näiden toimintojen käyttäminen on hyödyllistä, jos esimerkiksi halutaan tutkia paikannetun esineen ympärillä olevia huoneita tai kiinteitä mittalaitteita ja niiden sijainteja.

Kuva 24: Kerroksen kaikkien mittalaitteiden asettaminen näkyviin samanaikaisesti.

5.4 Peruskäyttäjän sovelluksen toteutuksen ongelmia ja ratkaisuja

Seuraavassa on esitelty neljä peruskäyttäjän sovelluksen toteutuksen vaikeinta kohtaa. Koska TUTWSN-mittausverkon tietokantaan ei saatu toteuttaa suoraa yhteyttä, täytyi väliin rakentaa komponentti, joka suoritti datan noutamisen *WSNExerciseAPI*-rajapinnan kautta. Peruskäyttäjän sovelluksessa graafisten objektien käsittely ja varsinkin graafisen objektin siirtäminen graafiselta alueelta toiselle tuottivat aluksi vaikeuksia. Esimerkiksi näkymän keskittäminen seurattavana olevaan kohteeseen tämän siirtyessä rakennuksesta ja / tai kerroksesta toiseen, ja toimenpiteen ilmaiseminen sovellusta käyttävälle henkilölle oli haastava toiminto toteuttaa.

5.4.1 TUTWSN-mittausverkon liikkuvien mittalaitteiden datan noutaminen peruskäyttäjän sovellukseen

Mittalaitteiden keräämän datan noutamiseen toteutettiin *WsnDataCollector*-sovellus C++:lla, joka käyttää *WsnExerciseAPI*-rajapintaa (katso kohdat 3.2 - 3.4 ja 5.5). Sen avulla kerätään TUTWSN-verkosta liikkuvien mittalaitteiden keräämiä tietoja, joita käytetään paikannusalgoritmissa. *WsnDataCollector* kerää datan csv-tiedostoon, joka ladataan *DownLoader*-luokan avulla. Lataus tapahtuu *QHttp*-luokan [37] funktioita käyttämällä. Tietojen hakemiseen käytettiin http-yhteyttä, koska suoraa linkkiä TUTWSN:n tietokantaan ei saatu. Seuraavassa on esitetty esimerkki latauksen suorittavasta ohjelmakoodista:

```
void Downloader::doDownload() {
```

```
  // Luodaan uusi QHttp-olio
  http = new QHttp(this);

  // Yhdistetään signaalit ja slotit
  connect(http, SIGNAL(stateChanged(int)), this,
  SLOT(stateChanged(int)));
  connect(http, SIGNAL(responseHeaderReceived(QHttpResponseHeader)),
  this, SLOT(responseHeaderReceived(QHttpResponseHeader)));
  connect(http, SIGNAL(requestFinished(int,bool)), this,
  SLOT(requestFinished(int,bool)));

  // Asetetaan isäntä ja haetaan sen alta osoitteesta
  // locationInfoDaily.csv –niminen tiedosto
  // [KÄYTTÄJÄTUNNUS] = valinnainen (lintulan tunnus)
  http->setHost("www.cs.tut.fi");
  http->get("/~[KÄYTTÄJÄTUNNUS]/locationInfoDaily.csv");
}
```

requestFinished(int, bool) -slotissa määritetään mihin kansioon ja tiedostoon *locationInfoDaily.csv* -tiedosto halutaan tallentaa:

```
void Downloader::requestFinished(int id, bool error) {
    if(error) {
        // Jos latauksessa ilmeni virhe
        qDebug() << "ERROR!";
    }
    else {
        // Muuten kaikki on OK ja voidaan luoda uusi QFile –tyyppinen
        // olio
        qDebug() << "OK";
        QFile *file = new QFile("Locations/wsnLocations.csv");

        // Jos tiedoston avaaminen onnistuu, kirjoitetaan sinne juuri
        // haetun tiedoston sisältö
        if(file->open(QFile::WriteOnly)) {
            file->write(http->readAll());
            file->flush();
            file->close();
        }
        delete file;
    }
}
```

5.4.2 Liikkuvien mittalaitteiden sijaintien reaaliaikainen päivittäminen käyttöliittymään

Peruskäyttäjän sovelluksen moottoriin on toteutettu ajastettu toiminto, joka lataa uusimmat liikkuvien mittalaitteiden naapurilaitetiedot kahden ja puolen minuutin välein. Ajastin laukaisee *doDownload()*-funktion kahden ja puolen minuutin välein, koska TUTWSN-verkon mittalaitteet lähettävät uutta tietoa viiden minuutin välein. Tällä tavoin käyttöliittymän näkymässä pyritään pitämään mahdollisimman reaaliaikaista tietoa mittalaitteiden sijainneista.

5.4.3 Mittalaitteen siirtyminen rakennuksesta ja kerroksesta toiseen käyttöliittymässä

Koska jokaista rakennuksessa olevaa kerrosta vastaa käyttöliittymäpuolella grafiikka-alue (*QGraphicsScene* [38]), johon liikkuvia mittalaitteita kuvaavat graafiset objektit piirretään, täytyy mittalaitetta kuvaava graafinen objekti poistaa sen vanhalta grafiikka-alueelta ja tämän jälkeen lisätä se uudelle. Ohjelmakooditasolla tämä on toteutettu yhdistämällä kaikki grafiikka-alueet signaalien avulla toisiinsa. Seuraavassa on esitetty ratkaisuesimerkki ohjelmakoodista:

```
// Ohjelmakoodi on MapNodeItem –olion funktiosta: updatePosition()
// Mittalaite on siirtynyt eri rakennukseen
if(pWsnInterface_->getMobileNodeMapIndex(this->getNodeID().toInt()) !=
mapIndex_)
{
    // Asetetaan mittalaitteen (graafinen objekti) sijainniksi oikeaa
    // rakennusta vastaava indeksinumero
    mapIndex_ = pWsnInterface_->
    getMobileNodeMapIndex(this->getNodeID().toInt());

    // Mittalaite on vaihtanut kerrosta
    if(pWsnInterface_->
    getMobileNodeFloorIndex(this->getNodeID().toInt()) != floorIndex_)
    {

        // Asetetaan mittalaitteelle uusi kerroksen kertova informaatio
        floorIndex_ = pWsnInterface_->
        getMobileNodeFloorIndex(this->getNodeID().toInt());
    }

    // Lähetetään signaali, jossa lähetetään grafiikkaobjektin
    // osoitin, rakennusta vastaava indeksinumero ja kerroksen
    // indeksinumero
    emit mapChanged(this, mapIndex_, floorIndex_);
```

Edellä lähetetyn signaalin vastaanottaa se *MapArea*-olio, jonka grafiikka-alueelle mittalaite siirretään:

```
void MapArea::addMobileWithNewLocation(MapNodeItem *item, int
mapIndex, int floorIndex) {

    if(mapIndex_ == mapIndex && floorIndex_ == floorIndex) {
    pScene_->addItem(item);
        ...
        ...
}
```

5.4.4 Esineen / mittalaitteen / alueen etsiminen hakusanalla ja grafiikka-alueen keskittäminen haluttuun kohteeseen

Kun hakusana on kirjoitettu *Finder*-olion (kuva 21) tekstikenttään ja tämän jälkeen painettu ”**Find**”-painiketta, lähettää *Finder*-olio *Mainwindow*-oliolle signaalin, joka vastaanoteteaan sen slotissa:

```
void MainWindow::searchItemsFromEngine(int type, QString keyWord) {

  // type 0 = Area name
  // type 1 = Area description
  // type 2 = Item name
  // type 3 = Item Id
  // type 4 = Node Id

  // Tyhjennetään Tuloslistaus vanhoista tiedoista
  pSearchListWidget_->clearList();

  switch (type)
    {
        ...
        ...
    case 2:

  // Asetetaan löydetyt esineet / laitteet / alueet listaan hakemalla
  // ne logiikkamoottorin puolelta
  pSearchListWidget_->addSearchResultsToList(pWsnInterface_->
  findItemsByName(keyWord));
  break;
  ...
  ...
}
```

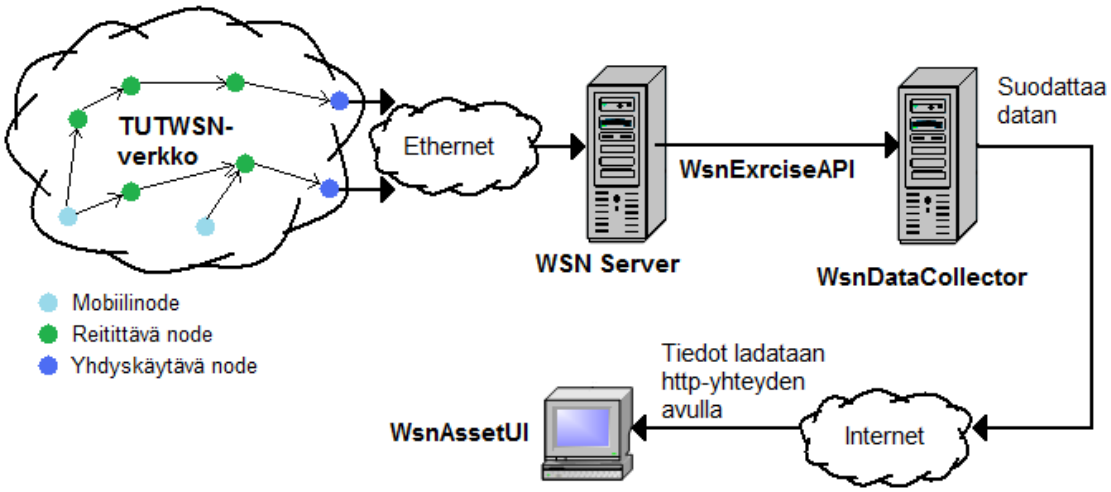
Moottorin puolella tapahtuva hakuoperaatio on seuraava:

```
QStringList WsnAssetLib::findItemsByName(QString keyWord) {

    // Käydään Qmap –tyyppinen tietorakenne läpi iteraattorin avulla
    // ja palautetaan SearchListWidgetille lista löydetyistä
    // tuloksista
    QStringList tmpList;
    QMap<long long, WsnItem*>::iterator it;

    for(it = wsnItems_.begin(); it != wsnItems_.end(); ++it) {
        QString tmpString = it.value()->getItemName();
        QString tmpKey = QString::number(it.key());

        // Jos moottorin tietorakenteessa oleva esine sisältää
        // hakusanan, se lisätään listaan
        if(tmpString.contains(keyWord)) {
            tmpList.push_back(tmpString);
            tmpList.push_back(tmpKey);
        }
    }
    return tmpList;
}
```

5.5 WsnDataColletor

WsnDataCollector on erillinen pieni C++ -sovellus, joka hakee *WsnExerciseAPI*-rajapinnan kautta TUTWSN-verkon dataa. Koska sensoriverkolta tuleva datan määrä on

suuri, on hyödyllistä suodattaa peruskäyttäjän sovellukselle vain tarpeellinen tieto. *WsnDataCollector* kerää tiedot vain liikkuvien mittalaitteiden lähettämistä naapurilaitetiedoista ja peruskäyttäjän sovellus osaa laskea liikkuvien mittalaitteiden sijainnin näiden tietojen perusteella. Kuvassa 25 on esitetty tiedon päätyminen TUTWSN-verkon yksittäiseltä mittalaitteelta peruskäyttäjän sovellukselle välivaiheineen. *WsnDataCollectorin* keräämät naapurilaitetiedot ovat liikkuvien mittalaitteiden keräämiä taulukoita, joissa joka toinen alkio on naapurilaitteen tunnistenumero ja joka toinen on vastaavan naapurilaitteen RSSI-arvo.



Kuva 25: Tiedon kulku TUTWSN-verkolta WsnDataCollectorin läpi peruskäyttäjän sovellukselle.

WsnDataCollector kerää tietoja kahteen eri csv-tiedostoon: *locationInfoDaily.csv* ja *locationInfoMonthly.csv*. Ensiksi mainittuun tiedostoon kerätään tietoja sadasta viimeksi rekisteröidystä tapahtumasta ja jälkimmäiseen kerätään TUTWSN-verkon liikkuvien mittalaitteiden naapurilaitetiedot viimeksi kuluneen kuukauden ajalta. *WsnDataCollector* voidaan muuttaa helposti keräämään tietoja vaikkapa viimeksi kuluneen 6 kuukauden ajalta, muuttamalla sen alussa määriteltyjen vakiomuuttujien arvoja. Peruskäyttäjän sovellus on myös suunniteltu niin, että *WsnDataCollector* voidaan vaihtaa käyttämään esimerkiksi TTY:n tietokonetekniikan laitoksella kehitettyä WSN openapia [39, 40, 41]. WSN openapi on kehitetty Qt Jambilla [42], joten sen avulla voitaisiin kerätä dataa TUTWSN-verkosta peruskäyttäjän sovellukselle. *WsnDataCollector* tuottaa peruskäyttäjän sovellukselle samassa csv-muodossa olevaa dataa, kuin esimerkiksi langattomien sensoriverkkojen sovellukset -kurssilla [6] käytettävän harjoitusrajapinnan (*WsnExerciseAPI*) [30] kautta saatava taulukkodata on.

6 TYÖN TULOKSENA SYNTYNEEN SOVELLUKSEN ARVIOINTI

Tässä luvussa esitellään langattoman sensoriverkon avulla toimivan omaisuudenhallintajärjestelmän käyttökokeumuksia. Aluksi esitellään, kuinka nopeasti pääkäyttäjän sovelluksen (editointityökalu) avulla voidaan asettaa langattoman sensoriverkon kattavan alueen huoneet ja kiinteät mittalaitteet karttapohjille, sekä liittää esine liikkuvan mittalaitteen kanssa. Tämän jälkeen esitetään peruskäyttäjän sovelluksen avulla suoritettuja paikannustuloksia ja kerrotaan kuinka tarkkoja ne olivat. Lopuksi kerrotaan valitun ohjelmointiympäristön etuja ja esitellään järjestelmän jatkokehitysajatuksia.

6.1 Pääkäyttäjän sovelluksen käyttötuloksia

Pääkäyttäjän sovelluksen käyttöliittymää testattiin suorittamalla sen avulla tehtäviä toimintoja ja mittaamalla sekuntikellolla toimintoihin kulunutta aikaa. Kerroksien karttapohjien päälle asetettiin yleensä vain ne huoneet, jotka sisälsivät kiinteitä mittalaitteita, poikkeuksina testitapaus, jossa mitattiin usean huoneen asettamiseen kulutettua aikaa ja seuraavassa mainittu tapaus.

Tietotalon ensimmäisen kerroksen kaikkien huoneiden (177 kpl) asetteluun kului 27 minuuttia ja 5 sekuntia ja niiden nimeämiseen kului 34 minuuttia ja 45 sekuntia. Asetteluun ja nimeämiseen yhteensä kulunut aika oli hieman päälle yksi tunti. Edellä mainittu mittaustulos saatiin, kun sovellusta testattiin ensimmäistä kertaa ja asettamalla useita huoneita saman karttapohjan päälle. Myöhemmissä testeissä sovellusta osattiin käyttää nopeammin, muun muassa kopioi ja liitä -toiminnon avulla. Taulukoissa 1 ja 3 on esitetty, kuinka paljon aikaa on kulunut huoneiden asetteluun eri rakennuksiin / kerroksiin pääkäyttäjän sovelluksen avulla. Taulukot 2 ja 4 kuvaavat kiinteiden mittalaitteiden asetteluun kulutettua aikaa. Huoneiden / alueiden asettelu tapahtui niin, että käytössä oli 2 näyttöä. Toisella näytöllä oli TTY:n tutka-palvelusta haetut karttapohjat, joista löytyvät huoneen / alueen tunnus ja kuvaus. Toisella näytöllä ajettiin pääkäyttäjän sovellusta. Mittalaitteiden asettelussa toisella näytöllä oli kartat, joista kiinteiden mittalaitteiden sijainnit löytyivät ja toisella näytöllä ajettiin pääkäyttäjän sovellusta.

Taulukko 1. Pääkäyttäjän sovelluksella suoritettu huoneiden asettelu ja nimeäminen, sekä toimintojen suorittamiseen kulunut aika rakennus- ja kerroskohtaisesti. Ainoastaan huoneet / alueet, joille mittalaitteita on asetettu, ovat lisätty sovelluksen avulla.

Rakennus ja kerros	*Huoneiden määrä, joissa mittalaitteita*	*Huoneiden asetteluun ja nimeämiseen kulunut aika (mm:ss)*
Päätalo 2. kerros	3	01:14
Sähkötalon kellari	8	03:15
Tietotalo 3. kerros	17	06:26
Tietotalo 1. kerros	23	08:40

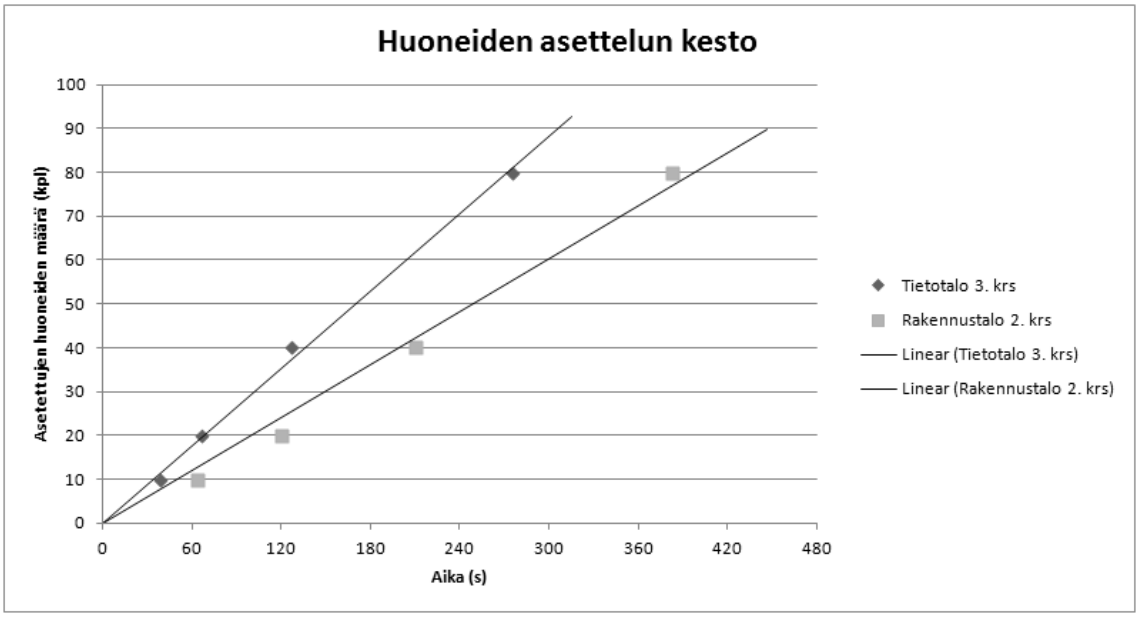
Taulukko 2. Pääkäyttäjän sovelluksella suoritettu mittalaitteiden asettelu ja siihen kulutettu aika rakennus- ja kerroskohtaisesti.

Rakennus ja kerros	*Mittalaitteiden määrä*	*Asetteluun kulunut aika (mm:ss)*
Sähkötalon kellari	13	01:31
Päätalo 2. kerros	15	02:17
Tietotalo 3. kerros	20	02:41
Tietotalo 1. kerros	32	05:53

Taulukko 3 ja kuva 26 esittävät tuloksia testistä, jossa mitattiin huoneiden sijoitteluun kulutettua aikaa. Testiin valittiin Tietotalon 3. kerros ja Rakennustalon 2. kerros. Aluksi karttapohjien päälle asetettiin 10 huonetta, sitten 20 huonetta. Kolmannella kierroksella asetettiin 40 huonetta ja lopuksi 80 huonetta. Samanmuotoisia alueita kopioitiin ja liitettiin karttapohjan päälle silloin, kun samanmuotoisia alueita havaittiin karttapohjalla. Testillä tutkittiin, nopeutuuko huoneiden asettelu kopioi ja liitä -toiminnon avulla, kun suuri määrä huoneita asetetaan karttapohjalle.

Taulukko 3. Pääkäyttäjän sovelluksella suoritettu huoneiden asettelu. Karttapohjana testissä olivat TTY:n Tietotalon 3.kerros ja Sähkötalon 2. kerros ja kaikki huoneet olivat suorakaiteen muotoisia.

Huoneiden määrä	*Kulunut aika (mm:ss) Tietotalo 3. kerros*	*Kulunut aika (mm:ss) Rakennustalo 2. kerros*
10	00:39	01:04
20	01:07	02:00
40	02:07	03:30
80	04:36	06:23



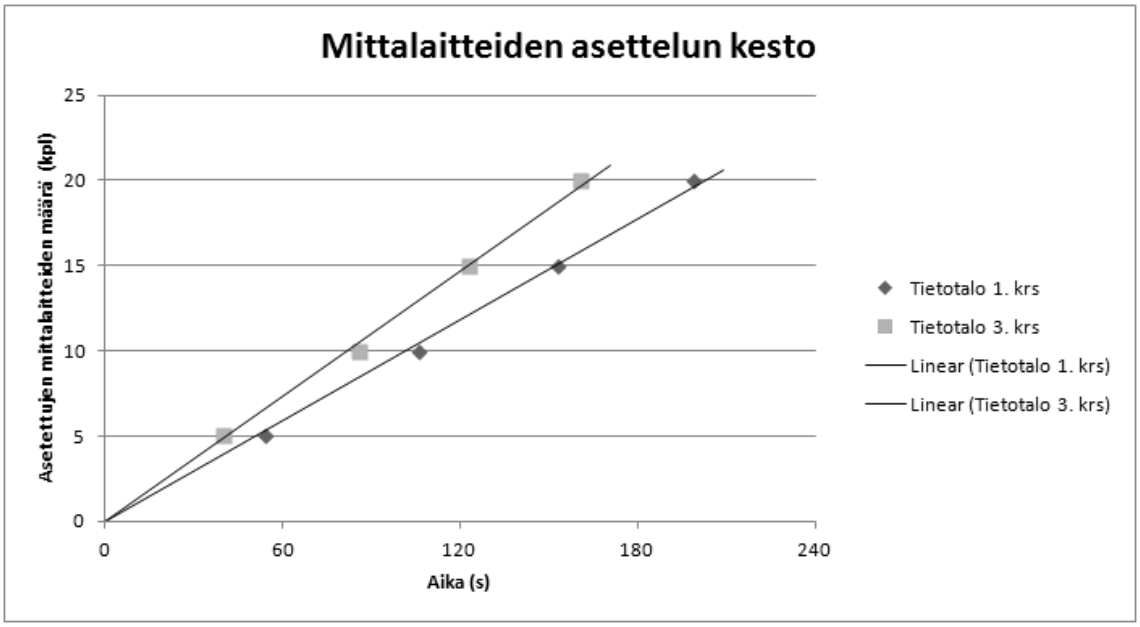
Kuva 26: Huoneiden sijoittelu ajan funktiona kahdessa eri tapauksessa.

Tuloksista voidaan sanoa, että kopio ja liitä -toiminnon käyttäminen ei nopeuta huoneiden asettelua niin paljon, että huoneita voitaisiin asetella eksponentiaalinen määrä kulutetun ajan suhteen. Huoneiden kopioiminen on hyödyllinen ominaisuus, mutta testin rakennusten pohjapiirrokset olivat arkkitehtuuriltaan sellaisia, että yli 10 huoneen liittäminen yhdestä kopiosta ei ollut mahdollista. Lisäksi huoneiden täsmällinen asettelu oikeaan kohtaan karttapohjalla vei suurimman osan ajasta.

Taulukossa 4 ja kuvassa 27 on esitetty tulokset mittalaitteiden lisäämisestä karttapohjan päälle pääkäyttäjän sovelluksen avulla. Mittalaitteita lisättiin ensin 5 kappaletta, sitten 10, 15 ja lopuksi 20 mittalaitetta ja ajat kirjattiin taulukkoon. Testi suoritettiin lisäämällä mittalaitteet Tietotalon ensimmäisen ja kolmannen kerroksen karttapohjille. Testin tarkoituksena on osoittaa, kuinka nopeasti sovelluksen avulla voidaan asettaa yhden kerroksen mittalaitteet järjestelmään.

Taulukko 4. Pääkäyttäjän sovelluksella suoritettu mittalaitteiden asettelu yhteen kerrokseen (Tietotalo 1. kerros ja Tietotalo 3. kerros).

Mittalaitteiden määrä	*Kulunut aika (mm:ss) Tietotalo 1. kerros*	*Kulunut aika (mm:ss) Tietotalo 3. kerros*
5	00:54	00:40
10	01:46	01:26
15	02:33	02:03
20	03:19	02:41



Kuva 27: Mittalaitteiden sijoittelu ajan funktiona kahdessa eri tapauksessa.

Mittalaitteiden lisääminen Tietotalon ensimmäiseen kerrokseen kesti kauemmin, koska se on pinta-alaltaan suurempi, kuin Tietotalon kolmas kerros. Testit osoittavat, että sovelluksen avulla kahdenkymmenen mittalaitteen asettaminen karttapohjalle kestää alle viisi minuuttia. Edellisten tulosten perusteella esimerkiksi koko TTY:n kattavan sensoriverkon mittalaitteiden asettaminen järjestelmään kestäisi alle tunnin.

6.2 Esineen paikantaminen peruskäyttäjän sovelluksen avulla

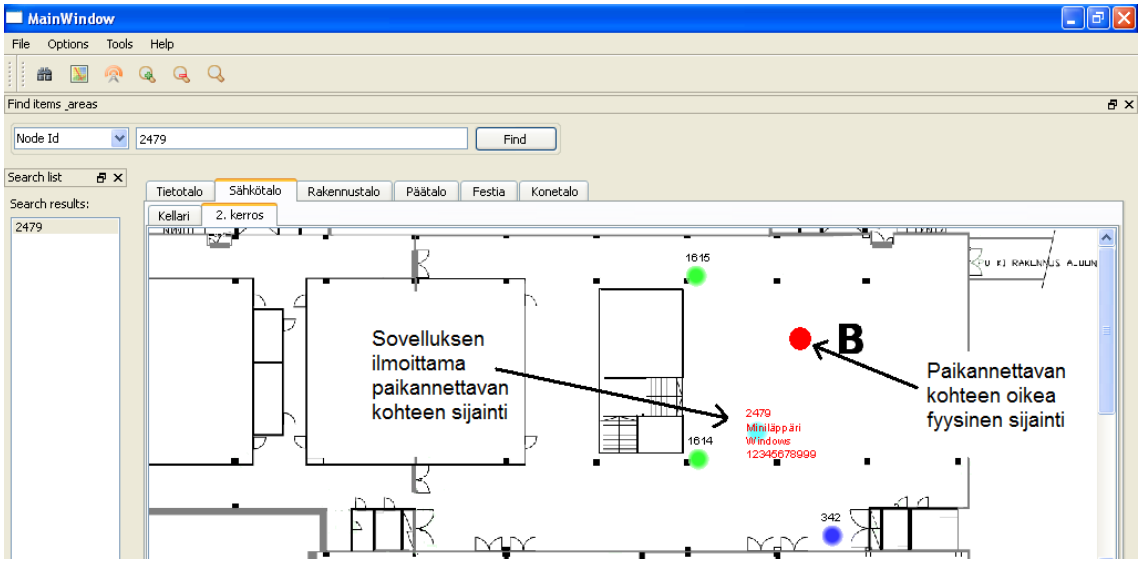
Seuraavissa kohdissa on esitelty muutamia tapauksia, joissa peruskäyttäjän sovelluksen avulla on paikannettu esine tai liikkuva mittalaite. Paikannettava kohde etsitään hakuehdoilla ja kohde esitetään peruskäyttäjän sovelluksen käyttöliittymässä. Paikantamiseen käytetään TUTWSN-mittausverkolta saatavien kiinteiden mittalaitteiden ilmoittamia RSSI-arvoja, joiden perusteella liikkuvan mittalaitteen todennäköinen sijainti voidaan laskea ja esittää käyttöliittymässä.

6.2.1 Testi 1: TUTWSN-verkon alueella kiertely liikkuvan mittalaitteen kanssa

Testi suoritettiin kulkemalla kannettavan tietokoneen kanssa TTY:n jokaisessa rakennuksessa, johon on asennettu kiinteitä TUTWSN-verkon mittalaitteita. Kannettavaan tietokoneeseen oli kiinnitetty liikkuva mittalaite (tunnistenumero: 2479). Kannettavassa tietokoneessa ajettiin samalla peruskäyttäjän sovellusta, jolla tarkkailtiin liikkuvan mittalaitteen sijainnin vaihtelua sovelluksen eri karttapohjien välillä. Jokaisen rakennuksen sisälle jäätiin odottamaan 5-10 minuuttia, jotta TUTWSN-verkolta saatiin päivittynyt tieto liikkuvan mittalaitteen uusista naapurilaitteista ja niiden RSSI-arvoista.

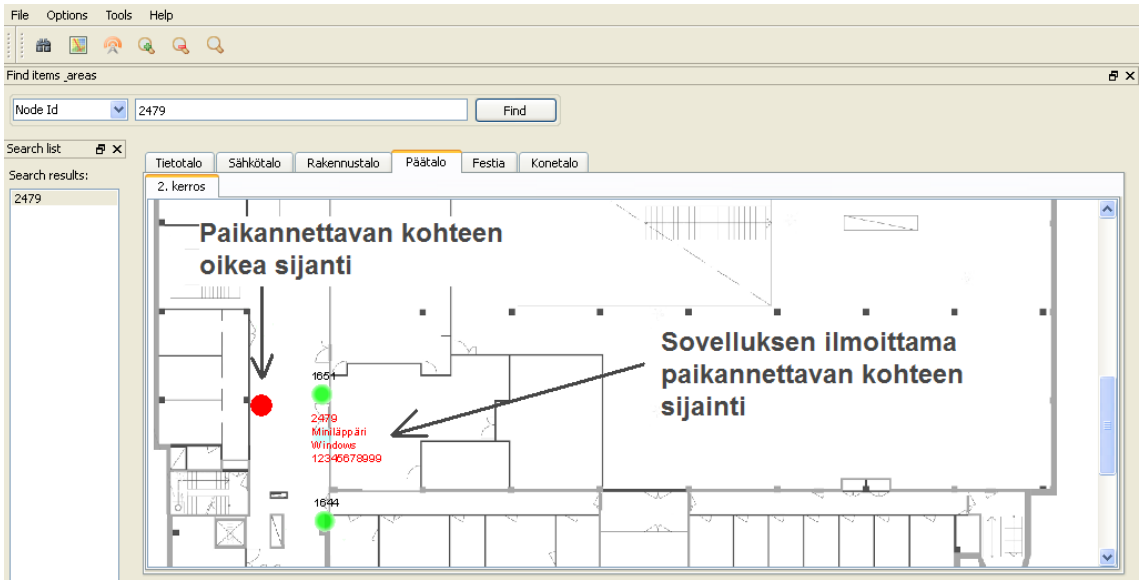
Tietojen latauduttua peruskäyttäjän sovellukseen, paikannettavan kohteen sijainti vaihtui eri karttapohjalta toiselle sen mukaan, missä rakennuksessa oltiin.

Kuvissa 28 ja 29 esiintyy tapaukset, jotka havainnollistavat peruskäyttäjän sovelluksen avulla suoritettua esineen paikantamista. Niissä on etsitty liikkuvaa mittalaitetta tunnistenumerolla 2479, jonka tiedetään olevan liitetty ”*Miniläppäri*”-nimiseen esineeseen. Mittalaitteen sijainti ilmoitetaan peruskäyttäjän sovelluksen käyttöliittymässä karttapohjan päällä. Mittalaite tunnistetaan karttapohjalla helposti, koska haun seurauksena se muuttuu valituksi ja aloittaa välkkymisen, sekä siihen liitetyn esineen kaikki tiedot näytetään mittalaitteen päällä. Lisäksi esineeseen liittyvät tekstit vaihtavat väriä, joten käyttäjän huomion tulisi kiinnittyä paikannettuun kohteeseen. Kuvassa 28 mittalaite on paikannettu Sähkötalon 2. kerrokseen. Kuvaan on merkitty punaisella ympyrällä mittalaitteen oikea sijainti Sähkötalossa. Vaaleansininen ympyrä, jonka päällä on punaista tekstiä esineen tiedoista, vastaa peruskäyttäjän sovelluksen ilmoittamaa mittalaitteen sijaintia.



Kuva 28: Ensimmäisessä paikannustestissä saatu esineen paikannustulos (vaaleansininen ympyrä) ja esineen oikea sijainti (punainen ympyrä). Vihreät ja tummansininen ympyrä ovat kiinteitä mittalaitteita, jotka liikkuva mittalaite on ilmoittanut naapurilaitteikseen viimeisimmän TUTWSN-verkolta saadun tiedon mukaan.

Kuvassa 29 mittalaite on paikannettu Päätalon 2. kerrokseen. Kuvassa liikkuvan mittalaitteen yläpuoleiselta naapurilaitteelta on saatu voimakkain RSSI-tieto. Edellisen tiedon perusteella peruskäyttäjän sovellus osaa sijoittaa liikkuvan mittalaitteen käyttöliittymässä lähimmäksi sitä kiinteää laitetta, jolta on saatu voimakkain RSSI-arvo. Jos naapurilaitteilta saadaan sama RSSI-arvo, asetetaan paikannettava kohde suoran puoliväliin, joka on vedetty naapurilaitteiden välille.



Kuva 29: Paikannustulos Päätalon 2. kerroksesta.

6.2.2 Testi 2: Langattoman mittalaitteen liittäminen postikärryyn

Testissä liitettiin langaton mittalaite TTY:n Tietotalon kerroksissa 1-4 liikkuvaan postikärryyn. Testin avulla tutkittiin kuinka paljon nopeammin postikärry löytyy, kun se etsitään peruskäyttäjän sovelluksen avulla verrattuna tapaukseen, jossa se etsitään käymällä Tietotalon kerrokset järjestelmällisesti kävellen läpi. Pahimmassa tapauksessa manuaalisesti etsimällä joudutaan käymään kaikissa Tietotalon kerroksissa ja etsittävä kohde löytyy vasta viimeisestä sivukäytävästä. Parhaassa tapauksessa kohde löytyy ensimmäiseltä käytävältä ja keskimääräisessä tapauksessa puolesta välistä etsittävää aluetta.

Yhden Tietotalon kerroksen läpikäyminen kävellen kestää 5 minuuttia, jos tutkitaan ainoastaan postikärryn kulkureitti (ei käydä huoneiden sisällä, tutkitaan vain käytävät). Koko Tietotalon läpikäyminen kestää 20 minuuttia, joten postikärry löytyy pahimmassa tapauksessa kahdenkymmenen minuutin etsinnän jälkeen. Keskimäärin postikärry löytyy kymmenessä minuutissa manuaalisen etsinnän jälkeen. Jos peruskäyttäjän sovellus on auki, ja postikärry paikoillaan, niin se löytyi kahdessa minuutissa. Tulokseen sisältyi sovellukseen käytetty hakuaika ja kävely paikannetun kohteen luokse.

Testi osoittaa, että yleisesti käytössä olevien esineiden (esimerkiksi laboratoriomittauslaite tai kannettava tietokone) etsimiseen kulutettu aika voitaisiin minimoida omaisuudenhallintajärjestelmän avulla. Arvioidaan, että Tietotalon alueelta etsittäisiin joka päivä 10 kertaa joitakin esineitä. Päivässä etsintöihin kulutettu aika olisi noin 100 minuuttia (1 tunti ja 40 minuuttia). Yhden työviikon aikana etsintöihin kuluisi 8 tuntia ja 20 minuuttia. Vuodessa etsintöihin kuluisi 433 tuntia ja 20 minuuttia (= noin

18 vuorokautta). Sovellusta käyttämällä samaan määrään etsintöjä kuluisi 3,6 vuorokautta.

Sovellusta käyttämällä voitaisiin etsintöihin kulutettu aika vähentää viidennekseen alkuperäisestä. Lisäksi järjestelmän avulla voitaisiin lähettää sähköposti tai tekstiviesti, kun postikärry kulkee tietyn huoneen tai alueen ohi (kerroksessa sijaitsevat postilokerikot). Tällä tavoin voitaisiin informoida, onko postikärry käynyt tietyn henkilön postilaatikon kohdalla, eikä tärkeiden postien takia tarvitsisi käydä tarkastamassa tilannetta jatkuvasti laatikolla.

6.2.3 Järjestelmän soveltuvuus omaisuudenhallintaan ja kohteiden paikantamiseen

Tutkituissa sensoriverkkoja hyödyntävissä omaisuudenhallintajärjestelmissä käytetään pääsääntöisesti RFID-tekniikkaa, tai viivakooditarroja, jotka tarvitsevat tuekseen erillisiä lukijoita [43]. Muutamassa sairaaloihin asennetuissa prototyyppisovelluksissa käytettiin mittalaitteita, jotka toimivat autonomisesti keskenään [44, 45]. Tässä työssä kehitetyn järjestelmän sensoriverkko (TUTWSN) toimii samalla tavalla.

Järjestelmän avulla paikannettavan kohteen viimeisin tunnettu sijainti saadaan selville muutaman minuutin sisällä. Tämä ei kuitenkaan tarkoita, että esine olisi samalla hetkellä siellä, koska esine voi olla liikkeessä juuri paikannushetkellä. Tarkastelemalla muutamia viimeksi tiedettyjä sijainteja voidaan päätellä onko paikannettava kohde paikallaan, vai liikkeessä. Järjestelmää testattaessa huomattiin, että paikannettavien kohteiden oikea fyysinen sijainti vastasi järjestelmän käyttöliittymässä ilmoitettua sijaintia noin 2-7 metrin tarkkuudella. Sijaintitiedon tarkkuus riippuu sensoriverkon mittalaitteiden määrästä ja sensoriverkon mittalaitteiden RSSI-tietojen tarkkuudesta.

Järjestelmä on mahdollista siirtää käyttämään jotain muuta, kuin TUTWSN-sensoriverkkoa. Editointityökalua varten tarvitaan ainoastaan monitoroitavan alueen rakennusten, kerrosten ja piha-alueiden pohjapiirrokset, joille sensoriverkko on asennettu ja kartta asennetuista sensoriyksiköistä. Peruskäyttäjän sovellukseen on tehtävä muutoksia mobiilisensoriyksiköiden paikkatietoa päivittävään funktioon, ellei uusi sensoriverkko tuota täysin samanlaisessa muodossa olevaa dataa, kuin TUTWSN-verkon *WsnExerciseAPI*. Eräs ratkaisu olisi luoda väliin adapteri-luokka, joka muuntaa datan peruskäyttäjän sovellukselle sopivaan muotoon.

Seuraavassa on esitetty positiivisia huomioita sovelluksen toiminnasta:

- 4 tunnin yhtäjaksoisen käytön jälkeen sovellus toimi moitteetta. Sovellusta ajettiin Qt:n debug-tilassa.
- Internet-yhteyden katkeaminen ei aiheuttanut sovelluksen kaatumista, vaan sovellus jatkoi toimintaansa normaalisti myös internet-yhteyden palattua.
- Sensoriverkon ulkopuolelle viety laite löysi verkon uudestaan noin 10 minuutin päästä, kun se tuotiin takaisin verkon alueelle (TUTWSN:n ominaisuus). Tämä huomattiin peruskäyttäjän sovelluksen käyttöliittymästä.
- Sovellus sopisi myös sellaisten henkilöiden käyttöön, joille uusi ympäristö on entuudestaan tuntematon ja heidän pitäisi löytää tietty huone nopeasti (sovelluksen avulla voidaan etsiä myös huoneita / alueita).

Sovelluksen rajoituksia ovat:

- Jos paikannettava laite on kerrosten välissä (portaikko), on vaikea päättää, kummalla karttapohjalla laite esitetään.
- Jos useissa eri kerroksissa olevat kiinteät mittalaitteet ilmoittautuvat naapurilaitteiksi ja ilmoittavat saman RSSI-arvon, on vaikea tehdä päätös, missä kerroksessa esine oikeasti on.
- Jos paikannettava laite on liikkeessä, uusi tieto laitteen sijainnista saadaan vasta viiden minuutin odottelun jälkeen (TUTWSN-mittausverkon ominaisuus).
- Jatkuvasti liikkeessä olevan laitteen paikantaminen on vaikeaa (TUTWSN-verkon ominaisuuksista johtuen).

6.3 Qt:n hyödyt ja haitat toteutuksessa

Qt-kehitysympäristön etuja ovat:

- Soveltuvuus olio-ohjelmointiin.
- Valmiit toteutukset tietyille asioille (esimerkiksi Qt Designer).
- Yksinkertaisen 2D-grafiikan esittäminen.
- Debug-tila (nopea kääntäminen, ei tarvita virtuaalipalvelimien pystyttämistä).
- Avoin lähdekoodi.
- Toimii useilla alustoilla (esimerkiksi Win, Mac OsX).
- Käyttöliittymiä tehdessä signals & slots -metodi on hyödyllinen.
- Paljon opeteltavaa (muun muassa useita valmiita luokkia).

Qt-kehitysympäristön haittoja ovat:

- Lopullista sovellusta ei voida ajaa esimerkiksi selaimessa.
- Jotkut asiat täytyi toteuttaa lähes tyhjästä (graafisen objektin koon ja muodon muuttaminen dynaamisesti).
- Paljon opeteltavaa.

6.4 Omaisuudenhallintajärjestelmän jatkokehitys

Pää- ja peruskäyttäjien sovellusten prototyypit soveltuvat nykyisessä muodossaan kohtuullisen hyvin TTY:n alueella liikkuvan omaisuuden monitoroimiseen. Sovelluksen täysi käyttöönottaminen uudessa ympäristössä vaatisi lisää virhetarkasteluja, suoran yhteyden sensoriverkon tietokantaan ja joidenkin toimintojen hiomista (esimerkiksi uusien erimuotoisten graafisten alueiden lisäämisen karttaeditoriin).

Tämän työn pohjalta syntynyttä prototyyppisovellusta käyttämällä ilmeni joukko jatkokehitysajatuksia:

- Integrointi videovalvontaan ja kulkukorttijärjestelmään.
- Sähköpostin ja / tai tekstiviestin lähettäminen, jos esine katoaa verkon sisältä.
- Kaksiulotteisen sijaintitiedon esittämisen sijaan kolmiulotteinen. Tämä vaatisi lähes kokonaan uuden sovelluksen toteuttamisen.

7 YHTEENVETO JA JOHTOPÄÄTÖKSET

7.1 Tulosten yhteenveto

Tässä diplomityössä tutkittiin langattomien sensoriverkkojen perusteita ja paikannustapoja. Esiteltiin itse toteutettu omaisuudenhallintajärjestelmä ja sen toteutukseen liittyvien ongelmien ratkaisuja. Omaisuudenhallintajärjestelmä koostui kahdesta erillisestä prototyyppisovelluksesta, joiden käyttöliittymien toteutukset olivat työn pääosassa. Paikannuksen apuna käytetty sensoriverkko oli TUTWSN-mittausverkko. Lopuksi esiteltiin tuloksia, joita sovelluksen avulla saavutettiin.

Pääkäyttäjän sovelluksen avulla sensoriverkon mittalaitteiden ja alueiden siirtäminen reaalimaailmasta sovelluksen käyttöliittymässä esitettyjen karttapohjien päälle on kohtuullisen nopeaa. Monia rakennuksia ja kerroksia, sekä satoja mittalaitteita käsittävän ympäristön siirtäminen digitaaliseen muotoon pääkäyttäjän sovelluksen avulla kestää muutaman tunnin.

Kohteiden paikannusta varten kehitetyn peruskäyttäjän sovelluksen arvioinnin perusteella kohteiden etsiminen hakusanan avulla on helppoa ja nopeaa. Kohteen paikannus ei riipu pelkästään peruskäyttäjän sovellukseen kehitetystä paikannusalgoritmista, vaan paikkatiedon tarkkuus riippuu myös TUTWSN-verkon sensorien mittaustuloksista. Työssä toteutettu sovellus on kuitenkin prototyyppi ja siihen kehitetty paikannusalgoritmi ei ole työn oleellisin osa. Peruskäyttäjän sovelluksen avulla kohteiden etsimiseen kulutettua aikaa voitaisiin vähentää viidesosaan verrattuna manuaaliseen etsintään kulutettuun aikaan.

Taulukko 5 kuvaa omaisuudenhallintajärjestelmän toteutuksen laajuutta. Siihen on merkitty luokkien ja ohjelmakoodirivien määrä jaoteltuna järjestemän eri osien mukaan. Taulukkoon on lisäksi listattu pää- ja peruskäyttäjien sovellusten käyttämien aputiedostojen nimet.

Taulukko 5. Yhteenveto omaisuudenhallintajärjestelmän osista.

	Pääkäyttäjän sovellus	*Peruskäyttäjän sovellus*	*WsnDataCollector*
Ajettavan binäärin nimi	WAAT.exe (WsnAssetAdminTool)	WSNAssetUI.exe	wsndatacollector
Kirjastotiedoston nimi	WsnAssetAdminLib.dll	WsnAssetLib.dll	-
Aputiedostojen määrä	5	4	2
Aputiedostojen nimet	defaultMap.csv, defaultNodeList.csv, areaList_default.csv, tkt_koneet_default.csv, nodeDefaultMap.csv	defaultMap.csv, defaultNodeList.csv, tkt_koneet_default.csv, nodeDefaultMap.csv	locationInfoDaily.csv, locationInfoMonthly.csv
Luokkia (moottori)	9	10	3
Luokkia (käyttöliittymä)	16	14	0
Koodirivejä (moottori)	2919	2937	207
Koodirivejä (käyttöliittymä)	4763	4735	0
Koodirivit yhteensä	7682	7672	207

7.2 Työn arviointi

Työ valmistui lähes aikataulun mukaan ja suurempia ongelmia työn suorittamisen aikana ei ilmaantunut. Projekti eteni pää- ja peruskäyttäjien sovellusten suunnittelusta niiden toteuttamiseen. Kun järjestelmän runko oli valmis, alkoi diplomityön kirjoittaminen. Loppuvaiheessa toteutettiin muutamia sovellusten vaatimia korjauksia ja suoritettiin sovellusten testaamista.

Työssä pääsi soveltamaan hyvin aikaisemmin suoritetuilta kursseilta opittuja asioita. Ohjelmistotekniikan laitoksen tarjoamien kurssien, erityisesti graafisen käyttöliittymän ohjelmoinnin ja käyttöliittymäsuunnittelun kursseilla läpikäydyistä asioista oli suuri hyöty tämän työn toteutusosuuden suorittamisen kannalta. Ohjelmistotuotannon ja langattomien sensoriverkkojen opintokokonaisuuksien yhdistäminen tässä työssä tuntui erittäin luontevalta ja työn suorittaminen oli mielenkiintoinen kokemus.

Työn toteutusosuutta tehdessä tuli opittua paljon uutta. Vaikka Qt:n tarjoamista mahdollisuuksista oli tiedossa paljon, niin työtä tehdessä tuli huomattua, että vielä on enemmän opittavaa. Monta uutta asiaa tuli opittua, jotka Qt:n avulla on mahdollista toteuttaa sovelluksia ohjelmoitaessa.

Työ tuntui onnistuneen hyvin, vaikka sovellusten toteutus olisi voinut olla hieman enemmän loppuun asti hiottu. Pää- ja peruskäyttäjän sovellukset kaipaisivat vielä joitain virhetarkasteluja ja testaukseen olisi pitänyt käyttää enemmän aikaa. Käytössä ei kuitenkaan ollut loputtomasti resursseja ja yhden ihmisen toteuttamaksi järjestelmäksi työn toteutusosuus oli melko laaja. Eniten työn suorittamista vaikeutti se, että vastaavanlaisia järjestelmiä ei ollut suuren yleisön saatavilla työtä kirjoittaessa.

LÄHTEET

[1] Tampere University of Technology, DACI Research Group, TUTWSN – Wireless Sensor Network, [Online]. Available: http://www.tkt.cs.tut.fi/research/daci/ra_tutwsn_overview.html, [2011, 12/20].

[2] Tampereen teknillinen yliopisto, Tietokonetekniikan laitos, TUTWSN-mittausverkon käyttö opetuksessa TTY:llä, [Online]. Available: http://www.tkt.cs.tut.fi/research/daci/mittausverkko/Mittausverkko-Opetus.pdf, [2011, 12/20].

[3] Tampereen teknillinen yliopisto, Tietokonetekniikan laitos, TUTWSN-mittausverkon toiminta, [Online]. Available: http://www.tkt.cs.tut.fi/research/daci/mittausverkko/toiminta.html, [2011, 12/20].

[4] M. Kuorilehto et. al "Ultra Low Energy Wireless Sensor Networks in Practise", Wiley, ISBN 978-0-470-05786-5.

[5] J. Jokinen, "Langattomien sensoriverkkojen toimintamatkan ja kattavuuden tutkimus," Insinöörintyö, Savonia-Ammattikorkeakoulu, Tekniikka, Kuopio, Joulukuu 2007.

[6] TKT-2301. Langattomien sensoriverkkojen sovellukset. TTY:n Tietokonetekniikan laitoksen järjestämä kurssi. Luentokalvot.

[7] R. Casas, A. Marco, J. L. Falco, H. Gracia, and J. I. Artigas, "DALMA – location aware alarm system for people with disabilities," pp. 744-751, 2006.

[8] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Distributed Computing Systems, 2005*. ICDCS 2005. *Proceedings*. *25th IEEE International Conference on*, pp 653-662, June 2005.

[9] C.-Y Chong and S. P. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol 91, no. 8, pp. 1247-1256, August 2003.

[10] A. Harter and A. Hopper, "A distributed location system for active office," in *Network*, IEEE, vol 8, no. 1, pp. 62-70, 1994.

[11] M. Terwilliger, A. Gupta, V. Bhuse, Z. Kamal, and M. Salahuddin, "A localization system using wireless network sensors: A comparison of two techniques," in *The Proceedings of the 1st Workshop on Positioning, Navigation and Communication – WPNC 04*, March 2004.

[12] M. Kotala, "Langattomat sensoriverkot," Opinnäytetyö, Tampereen Ammattikorkeakoulu, Tampere, Suomi, Toukokuu 2010.

[13] J. Suhonen, M. Kohvakka, M. Hännikäinen, and T. D. Hämäläinen, ”Design, implementation, and experiments on outdoor deployment of wireless sensor network for environmental monitoring,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation, 6th International Workshop, SAMOS 2006*, ser. Lecture Notes in Computer Science, vol 4017.

[14] Center for Transportation Research and Education. Governmental Accounting Standards Board Statement No. 34, [Online]. Available: http://www.ctre.iastate.edu/gasb34/intropart1.pdf/ , [2011, 11/21].

[15] L. Huhtala & J. Leinonen, ”Omaisuudenhallintakäyttöliittymä langattomille sensoriverkoille,” Kandidaatin työ, Tampereen Teknillinen Yliopisto, Tietokonetekniikan laitos. 27 s. Toukokuu 2010.

[16] L. Huhtala & J. Leinonen, ”Omaisuudenhallintajärjestelmän määrittelydokumentti,” v. 2.3, 41 s. Kesäkuu 2010.

[17] RFID Lab Finland ry, ”FiHTA_esitelm_RFIDLab-2.pdf,” Seminaarikalvot.

[18] H. Liu, H. Darabi, P. Banerjee and J. Liu, “Survey of Wireless Indoor Positioning Techniques and Systems,” *IEEE Transactions on systems, man, and cybernetics—part C: Applications and reviews*, vol. 37, no. 6, November 2007, pp. 1067-1080.

[19] A. Smith, H. Balakirshnan, M. Goraczko, and N. Priyantha, ”Tracking moving devices with the cricket location system,” in *MobiSys ’04: Proceedings of the 2nd international conference on Mobile systems, applications, and services,* pages 190–202, New York, NY, USA, 2004. ACM Press.

[20] A. Kotanen, ”Positioning with short-range radio networks,” Master's thesis, Tampere University of Technology, Tampere, Finland, August 2003.

[21] I. Kortelainen, ”ZigBee-paikannus,” Opinnäytetyö, Lahden Ammattikorkeakoulu, Kevät 2010.

[22] N. Patwari, J. N. Ash, S. Kyperountas, A. O. Hero III, R. L. Moses, and N. S. Correal, ”Locating the nodes: cooperative localization in wireless sensor networks,” *Signal Processing Magazine*, IEEE, vol. 22, no. 4, pp. 54-69, 2005.

[23] E. Elnahrawy, X. Li, and R. P. Martin, ”The limits of localization using signal strength: a comparative study,” in *Sensor and Ad Hoc Communications and Networks, 2004*. IEEE SECON 2004. *2004 First Annual* IEEE *Communications Society Conference on*, 2004, pp. 406-414.

[24] M. A. Al-Nuaimi, R. M. Shubair and K. O. Al-Midfa, “Direction of arrival estimation in wireless mobile communications using minimum veriance distorsionless response,” *The Second International Conference on Innovations in Information Technology (IIT’05)*, 2005.

[25] R. Roberts, “TDOA Localization Techniques,” *Project: IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)*, October 2004.

[26] J. Hightower and G. Borriello, ”Location sensing techniques,” Computer Science and Engineering, University of Washington, 2001.

[27] Lintula. Tampereen teknillisen yliopiston Tieto- ja sähkötekniikan tiedekunnan alainen Unix/Linux -ympäristö, [Online]. Available: http://www.cs.tut.fi/lintula, [2012, 02/22].

[28] I. Haikala & J. Märijärvi, ”Ohjelmistotuotanto,” Talentum 2006. pp 101-157. ISBN 952-14-0850-2.

[29] Nokia Corporation. Qt Reference Documentation. Qt version 4.7, [Online]. Available: http://doc.qt.nokia.com/4.7/, [2011, 12/12].

[30] TKT-2301 Exercise API, July 2010, [Online]. Available: http://www.tkt.cs.tut.fi/kurssit/2301/S11/material/wsnapi.pdf, [2011, 12/12].

[31] Vesanen, A. *Dynaamisesti linkitettävät kirjastot.* Oulun yliopisto, Tietojenkäsittelytieteiden laitos. 2008, [Online]. Available: http://www.tol.oulu.fi/users/ari.vesanen/windowsohjelmointi/Luennot/WO_DLL.html , [2011, 11/16].

[32] M. Rintala & J. Jokinen, ”Olioiden ohjelmointi C++:lla,” Talentum 2005. pp 27-50. ISBN 952-14-0936-3.

[33] Nokia Corporation. Qt 4.7: QGraphicsItem Class Reference. [Online], Available: http://doc.qt.nokia.com/4.7/qgraphicsitem.html, [2011, 12/16].

[34] Wolfram MathWorld. Rotation Matrix, [Online]. Available: http://mathworld.wolfram.com/RotationMatrix.html, [2011, 10/08].

[35] S. Owen, 2D Rotation, June 1999, [Online]. Available: http://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/2drota.htm, [2011, 10/09].

[36] Nokia Corporation. Qt 4.7: QStringList Class Reference. [Online], Available: http://doc.qt.nokia.com/4.7/qstringlist.html, [2011, 12/16].

[37] Nokia Corporation. Qt 4.7: QHttp Class Reference. [Online], Available: http://doc.qt.nokia.com/4.7/qhttp.html, [2012, 02/01].

[38] Nokia Corporation. Qt 4.7: QGraphicsScene Class Reference. [Online], Available: http://doc.qt.nokia.com/4.7/qgraphicsscene.html, [2012, 02/01].

[39] J. Peuralahti, J. Nurmilaakso, O. Mahlamäki, M. Vuori, O. Kivelä, M. Rentto ja M. Hännikäinen, ”wsn openapi - Sensoriverkon avoimet rajapinnat,” *WIREPAS-projekti*, 11 s.

[40] Tampere University of Technology, DACI Research Group, M. Rentto, Sensor Archive Data Format (SADF), [Online]. Available: http://openapi.elisalabs.fi:443/rfc/rfc_sadf.html, [2011, 12/20].

[41] J. Nurmilaakso & J. Peuralahti, Sensor Data Communication Using SIMPLE, [Online]. Available: http://openapi.elisalabs.fi:443/rfc/rfc_sdc_using_simple.html, [2011, 12/20].

[42] Nokia Corporation. Qt Jambi Reference Documentation. Qt Jambi version 4.5.2_01, [Online]. Available: http://doc.qt.nokia.com/qtjambi-4.5.2_01/com/trolltech/qt/qtjambi-index.html, [2011, 12/20].

[43] TVL Inc. , *WiseTrack is Asset Management for Tracking Assets using RFID, Bar Code, Mobile Scanners and the Web* [Homepage of WiseTrack], [Online]. Available: http://www.wisetrack.com/, [2012, 01/07].

[44] K. Sun-Jin, K. Sun-Joong, J.H. Seo & Krishna, J. ”Wireless Sensor Network based Asset Tracking Service,” *Management of Engineering & Technology, Portland International Conference*, 27-31 July 2008. pp. 2643-2647.

[45] TUTKAIN: Langattoman sensoriverkon pilottikäyttö Kainuun keskussairaalassa, Tampereen teknillinen yliopisto, Tietokonetekniikan laitos, TUTKAIN henkilöpaikannus- ja henkilöturvajärjestelmä, [Online]. Available: http://www.tkt.cs.tut.fi/research/daci/mittausverkko/Mittausverkko-Henkiloturva.pdf, [2011, 12/20].

LIITE 1: WSNASSETADMINLIB–KIRJASTON RAJAPINTAFUNKTIOT

```
//! Returns an error message
virtual QString getErrorMessage() = 0;

//! Adds new floor to map's data structure with given name
virtual void addFloorToMap(const QString floorName) = 0;

//! Returns number of floors in map with given index number
virtual int getFloors(const int mapIndex) = 0;

//! Returns number of maps (how many maps there is in data structure)
//! Maps represents buildings in this application (map can be like:
//! Tietotalo, Konetalo, Päätalo...)
virtual int getMaps() = 0;

//! Returns index number of map with given name
virtual int getMapIndex(const QString mapName) = 0;

//! Returns floor's index number with given map index and floor name
virtual int getFloorIndex(const int mapIndex, const QString floorName)
= 0;

//! Returns number of rooms with given map and floor indexes
virtual int getRooms(const int mapIndex, const int floorIndex) = 0;

//! Returns map’s picture path of specified map index and floor
virtual QString getMapPicture(const int mapIndex, const int
floorIndex) = 0;

//! Returns name of the floor with given parameters (index numbers of
//! map and floor)
virtual const QString getFloorName(const int mapIndex, const int
floorIndex) = 0;

//! Returns name of the map with given index number
virtual const QString getMapName(const int index) = 0;

//! Adds new map to data WsnAssetAdminLib's structure (created when
//! new map is added in GUI side)
virtual void addMap(const QString mapName) = 0;

//! Adds specified floor to map (when new floor is created in GUI
//! side)
virtual void addFloorToMap(const QString mapName, const QString
mapPicturePath, const QString floorName) = 0;

//! Function checks that map file exists and is in right format
virtual bool isMapFile(const QString mapFileName) = 0;
//! Function checks that default map exists
virtual bool isDefaultMap() = 0;

//! Saves all the information of buildings, floors, rooms, nodes in
//! rooms etc. into a .csv file
virtual bool saveMapToFile() = 0;

//! Loads map file with given file name
virtual bool loadMapFile(QString fileName) = 0;
```

```
//! Creates new map if given file name doesn't exist already
virtual bool newMapFile(QString filename) = 0;

//! Parses CSV file (Contains information about map, floors rooms
//! etc.)
virtual bool parseCSVFile(const QString fileName) = 0;

//! Parses area / node / item listing with given file name parameter
virtual bool parseAreaListFromCSV(const QString fileName) = 0;
virtual bool parseNodeListFromCSV(const QString fileName) = 0;
virtual bool parseItemListFromCSV(const QString fileName) = 0;
virtual bool parseMapNodesFromCSV(const QString fileName) = 0;

//! Clears all data from data structures in wsnAssetAdminLib (used for
//! example when new map / load map button is pressed in GUI side)
virtual void clearOldData() = 0;

//! Sets map's index number to data structure for later use (index
//! number comes from GUI part and it is needed when map is being
//! saved)
virtual void setMapIndex(const int index, const QString name) = 0;

//! Adds room to data structure when graphical room item is dropped on
//! map with given mapIndex
virtual void addRoomToFloor(int mapIndex, int floorIndex) = 0;

//! Sets room's information (data comes from GUI and it is stored to
//! WsnAssetLib's data structures) when map is being saved
virtual void setRoomInformation(int mapIndex, int floorIndex, int
roomIndex, const QString name, const QString description, const char
shape) = 0;

//! Sets known GraphicsScene position (topleft coordinate) into room's
//! private data variable. If room is polygon shaped then all of the
//! corner positions of room are also stored
virtual void setRoomPos(int mapIndex, int floorIndex, int roomIndex,
QPointF pos) = 0;
virtual void setPolygonRoomPoints(int mapIndex, int floorIndex, int
roomIndex, QPointF tpLeft, QPointF btLeft, QPointF btRight, QPointF
tpRight) = 0;

//! Sets room's width/height to room's data structure
virtual void setRoomWidth(int mapIndex, int floorIndex, int roomIndex,
const qreal width) = 0;
virtual void setRoomHeight(int mapIndex, int floorIndex, int
roomIndex, const qreal height) = 0;

//! Sets Area's (GarphicsSceneItem) rotation (0-360 degrees) into
//! room's private variable
virtual void setRoomRotation(int mapIndex, int floorIndex, int
roomIndex, qreal angle) = 0;

//! Returns information of room with given parameters
virtual const QString getRoomName(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QString getRoomDescription(int mapIndex, int floorIndex,
int roomIndex) = 0;
virtual const QString getRoomShape(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomPos(int mapIndex, int floorIndex, int
roomIndex) = 0;
```

```
virtual const QPointF getRoomTopLeft(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomBotLeft(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomBotRight(int mapIndex, int floorIndex,
int roomIndex) = 0;
virtual const QPointF getRoomTopRight(int mapIndex, int floorIndex,
int roomIndex) = 0;
virtual const qreal getRoomWidth(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const qreal getRoomHeight(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const qreal getRoomRotation(int mapIndex, int floorIndex, int
roomIndex) = 0;

//! Adds node object to room's data structure (when node is dropped on
//! room in GUI side)
virtual void addRoomNode(int mapIndex, int floorIndex, int roomIndex,
int nodeIndex, QPointF pos, QString color, long long nodeItemID,
QString itemBoundTime) = 0;

//! Returns number of nodes in one room with given parameters
virtual const int getRoomNodes(int mapIndex, int floorIndex, int
roomIndex) = 0;

//! Returns ID number / type / position on map / color of node with
//! given parameters
virtual const int getNodeID(int mapIndex, int floorIndex, int
roomIndex, int nodeIndex) = 0;
virtual const QString getNodeType(int nodeID) = 0;
virtual const QPointF getNodePos(int mapIndex, int floorIndex, int
roomIndex, int nodeIndex) = 0;
virtual const QString getNodeColor(int mapIndex, int floorIndex, int
roomIndex, int nodeIndex) = 0;

//! Returns item id number / time when item has been combined with
//! node with given parameters
virtual const long long getNodeItemId(int mapIndex, int floorIndex,
int roomIndex, int nodeIndex) = 0;
virtual const QString getItemBoundTime(int mapIndex, int floorIndex,
int roomIndex, int nodeIndex) = 0;

//! Removes room from data structure with given indexes
virtual void removeRoom(int mapIndex, int floorIndex, int roomIndex) =
0;

//! Returns list of areas / nodes / items (Data can be placed for
//! example into QListWidget in GUI side)
virtual QStringList getAreaNameList() = 0;
virtual QStringList getNodeList() = 0;
virtual QStringList getItemList() = 0;

//! Returns information of item with given ID number
virtual const QString getItemName(long long itemID) = 0;
virtual const QString getItemLocation( long long itemID) = 0;
virtual const QString getItemType(long long itemID) = 0;
virtual const QString getItemMark(long long itemID) = 0;
virtual const QString getItemModel(long long itemID) = 0;
virtual const QString getItemOwner(long long itemID) = 0;
//! Clears data structures which contains info of WsnRoom objects
virtual void clearRooms() = 0;
```

LIITE 2: WSNASSETLIB –KIRJASTON RAJAPINTAFUNKTIOT

```
//! Returns an error message
virtual QString getErrorMessage() = 0;

//! Returns number of floors in map with given index number
virtual int getFloors(const int mapIndex) = 0;

//! Returns number of maps (how many maps there is in data structure)
//! Maps represents buildings in this application (map can be like:
//! Tietotalo, Konetalo, Päätalo...)
virtual int getMaps() = 0;

//! Returns number of rooms with given map and floor indexes
virtual int getRooms(const int mapIndex, const int floorIndex) = 0;

//! Returns map picture path of specified map index and floor
virtual QString getMapPicture(const int mapIndex, const int
floorIndex) = 0;

//! Returns name of the floor with given parameters (index numbers of
//! map and floor)
virtual const QString getFloorName(const int mapIndex, const int
floorIndex) = 0;

//! Returns name of the map with given index number
virtual const QString getMapName(const int index) = 0;

//! Parses CSV file (Contains information about map, floors rooms
//! etc.)
virtual bool parseCSVFile(const QString fileName) = 0;

//! Parses area / node / item listing with given file name parameter
virtual bool parseAreaListFromCSV(const QString fileName) = 0;
virtual bool parseNodeListFromCSV(const QString fileName) = 0;
virtual bool parseItemListFromCSV(const QString fileName) = 0;
virtual bool parseMapNodesFromCSV(const QString fileName) = 0;

//! Clears all data from data structures in wsnAssetLib (used for
//! example when load map button is pressed in GUI side)
virtual void clearOldData() = 0;

//! Returns information of room with given parameters
virtual const QString getRoomName(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QString getRoomDescription(int mapIndex, int floorIndex,
int roomIndex) = 0;
virtual const QString getRoomShape(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomPos(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomTopLeft(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomBotLeft(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const QPointF getRoomBotRight(int mapIndex, int floorIndex,
int roomIndex) = 0;
virtual const QPointF getRoomTopRight(int mapIndex, int floorIndex,
int roomIndex) = 0;
```

```
virtual const qreal getRoomWidth(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const qreal getRoomHeight(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const qreal getRoomRotation(int mapIndex, int floorIndex, int
roomIndex) = 0;

//! Returns room's map / floor index number
virtual const int getRoomsMapIndex(int mapIndex, int floorIndex, int
roomIndex) = 0;
virtual const int getRoomsFloorIndex(int mapIndex, int floorIndex, int
roomIndex) = 0;

//! Returns number of nodes in one room with given parameters
virtual const int getRoomNodes(int mapIndex, int floorIndex, int
roomIndex) = 0;

//! Returns ID number / type / position on map / color of node with
//! given parameters
virtual const int getNodeID(int mapIndex, int floorIndex, int
roomIndex, int nodeIndex) = 0;
virtual const QString getNodeType(int nodeID) = 0;
virtual const QPointF getNodePos(int mapIndex, int floorIndex, int
roomIndex, int nodeIndex) = 0;
virtual const QString getNodeColor(int mapIndex, int floorIndex, int
roomIndex, int nodeIndex) = 0;

//! Returns item id number / time when item has been combined with
//! node with given parameters
virtual const long long getNodeItemId(int mapIndex, int floorIndex,
int roomIndex, int nodeIndex) = 0;
virtual const QString getItemBoundTime(int mapIndex, int floorIndex,
int roomIndex, int nodeIndex) = 0;

//! Returns list of areas / nodes / items (Data can be placed for
//! example in to QListWidget in GUI side)
virtual QStringList getAreaNameList() = 0;
virtual QStringList getNodeList() = 0;
virtual QStringList getItemList() = 0;

//! Returns information of item with given ID number
virtual const QString getItemName(long long itemID) = 0;
virtual const QString getItemLocation(long long itemID) = 0;
virtual const QString getItemType(long long itemID) = 0;
virtual const QString getItemMark(long long itemID) = 0;
virtual const QString getItemModel(long long itemID) = 0;
virtual const QString getItemOwner(long long itemID) = 0;

//! Clears data structures which contains info of WsnRoom objects
virtual void clearRooms() = 0;

//! Returns list of areas / items / nodes that are filtered from data
//! structure with given keyword
virtual QStringList findAreasByName(QString keyWord) = 0;
virtual QStringList findAreasByDescription(QString keyWord) = 0;
virtual QStringList findItemsByName(QString keyWord) = 0;
virtual QStringList findItemsById(QString keyWord) = 0;
virtual QStringList findNodesById(QString keyWord) = 0;
```

```
//! Starts timer that triggers updateLocations() funtion (updates
//! mobile node locations once in every 5 minutes)
virtual void startTrackingMobiles() = 0;

//! Gets mobile nodes locations during last month
virtual void getMonthlyLocations() = 0;
```

LIITE 3: ESIMERKKI TALLENNETUSTA KARTTATIEDOSTOSTA

Taulukko 6. Alkuosa tallennetusta csv-tyyppisestä karttatiedostosta.

Karttatiedoston nimi	*Karttatiedoston GPS-koordinaatti*	*Rakennuksen nimi*	*Kerroksen nimi*	*Huoneen nimi*	*Huoneen kuvaus*
Images/tietotalo1.png	0, 0	Tietotalo	1. kerros	TB104	Luentosali
Images/tietotalo2.png	0, 0	Tietotalo	2. kerros	TA203	TATK-luokka
Images/sahkokellari.png	20, 5	Sähkötalo	Kellari	Telok	Kerhohuone

Taulukko 7. Loppuosa tallennetusta csv-tyyppisestä karttatiedostosta.

*Huoneen muodon tyyppi**	*Huoneen x-kulmakoordinaatti*	*Huoneen y-kulmakoordinaatti*	*Huoneen leveys*	*Huoneen korkeus*	*Huoneen rotaatiotieto*
R	2080	1228	537	343	0
C	3139	713	324	230	-16
R	110	208	205	191	0

*Jos huoneen muodon tyyppi on P (Polygon), jätetään leveys- ja korkeusinformaatio pois ja csv-tiedostoon lisätään vielä seuraavat kentät ennen huoneen rotaatiotietoa:

- Huoneen 1. x-kulmakoordinaatin arvo
- Huoneen 1. y-kulmakoordinaatin arvo
- Huoneen 2. x-kulmakoordinaatin arvo
- Huoneen 2. y-kulmakoordinaatin arvo
- Huoneen 3. x-kulmakoordinaatin arvo
- Huoneen 3. y-kulmakoordinaatin arvo
- Huoneen 4. x-kulmakoordinaatin arvo
- Huoneen 4. y-kulmakoordinaatin arvo

LIITE 4: ESIMERKKI TALLENNETUSTA MITTALAITETIEDOSTOSTA

Taulukko 8. csv-tyyppinen tiedosto karttapohjalle tallennetuista mittalaitteista.

Huoneen tunnus, johon mittalaite on yhdistetty	*Mittalaitteen tunnistenumero*	*Mittalaitteen x-koordinaatti*	*Mittalaitteen y-koordinaatti*	*Mittalaitteen väritieto*	*Mittalaitteeseen yhdistetyn esineen tunnistenumero* *
TB104	1687	2530	1265	#00ff00	-1
TB104	505	2553	1532	#0000ff	-1
TB207	50	2802.73	1391.6	#00ff00	-1
TA201	1750	3344.73	996.094	#00ff00	-1

*Mittalaitteeseen yhdistetyn esineen tunnistenumeron arvo on -1, jos mittalaitteeseen ei ole yhdistetty mitään esinettä.

LIITE 5: ESIMERKKI LADATTAVASTA ALUELISTAUKSESTA

Taulukko 9. csv-tyyppinen tiedosto alueiden tunnuksista.

Huoneen tunnus
TB103
TB104
TB207
TA201
TC220
SB105
FA030
K1704

LIITE 6: ESIMERKKI LADATTAVASTA MITTALAITETIEDOSTOSTA

Taulukko 10. csv-tyyppinen tiedosto mittalaitteiden tiedoista.

Mittalaitteen tunnistenumero	*Mittalaitteen tyyppi*	*Hälytysraja-arvo patterien jännitteelle*	*Napin A tyyppi*	*Napin B tyyppi*
1953	router	2.8	hoitajakutsuA	hoitajakutsuB
1687	router	2.8	hoitajakutsuA	hoitajakutsuB
2520	router	2.8	hoitajakutsuA	hoitajakutsuB
506	GW	2.8	hoitajakutsuA	hoitajakutsuB
2465	mobiili	2.8	hoitajakutsuA	hoitajakutsuB
2460	mobiili	2.8	hoitajakutsuA	hoitajakutsuB

LIITE 7: ESIMERKKI LADATTAVASTA ESINETIEDOSTOSTA

Taulukko 11. csv-tyyppinen tiedosto esineiden tiedoista.

Tyyppi	*Nimi*	*Merkki*	*Malli*	*Alkuperäinen sijainti*	*Omistaja*	*Laiterekisterinumero*
LintulaLinux	jokukone	IBM	x3650 M2, 7947-22G	TC420	TKT	12345678901
Sun	jokutoinen kone	Sun	SunFire V880	TC420	TKT	12345678902
Tulostin	tc315	HP	Laserjet 4050N	TC315	TKT	12345678903
Tulostin	tkt-c3-color	HP	color LaserJet 4600	TC3aul	TKT	12345678904
Windows	jokukolmaskone	HP	dc7100 CMT	TG324	TKT	12345678905